BBM 104 – Introduction to Programming II Lab.
Programming Assignment 2 – Object-Oriented Programming with Java
Developing a simple mini online shopping application – HUBBM Bazaar[1]

Developed for the Spring 2017 term by Research Assistants Selim Yılmaz and Selma Dilek

Submission:

- **Eclipse Project Folder Zipped as b<StudentID>.zip**
- **Deadline: 12.04.2017 23:59**

Table of Contents

1 INTRODUCTION

Object-oriented programming (OOP) is more than just adding a few new features to programming languages. It is rather a new *way of thinking* about decomposing and modeling problems with less complexity and more code reuse when developing programming solutions. In OOP, a program is viewed as a collection of loosely connected objects, each of which is responsible for specific tasks. It is through the interaction of these objects that computation proceeds and the model works as a whole.

[1] Covered subjects: OOP Basics, Abstraction, Encapsulation, Inheritance, Polymorphism, UML class diagrams, Javadoc

In this assignment, you will work with the concepts of OOP in order to practice them and observe their advantages. By the end of this assignment, you will learn the concepts of relationships among classes, encapsulation, abstraction, inheritance and polymorphism, as well as how to and why use Javadoc in your Java projects.

1.1 LEARNING OBJECTIVES

The learning objective of this programming assignment is to let students practice the following concepts of Java programming language in particular (and OOP in general):

- Object-oriented programming
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- UML class diagrams
- Javadoc

1.2 MARKING SCHEME

The marking scheme is shown below. Carefully review each mark component.

Component	Mark %
Creation of Base Classes with the requested hierarchical structure (the main class of your application must be named **`Main.java`**)	30%
Successful instantiation of all user and item objects from the input files	15%
Correct execution of events and actions as requested in the input file with commands	40%
Drawing of UML class diagrams saved as **`uml.jpg`**	5%
Javadoc that gives a detailed description of the classes in **`javadoc`** folder	10%
Total	**100%**

Note: In your solutions, your algorithm and code should be as simple as possible.

2 USEFUL INFORMATION

In this section you can find some useful beginner's level information that you will need for this project. For more information on each subject you need to do additional research and consult other resources (e.g. lecture notes, textbook, Internet resources, etc.).

2.1 ABSTRACTION

Abstraction is a general concept that denotes the progress of modeling "real things" into programming language. For example, when you write a class named Person, you abstract a real person into a type (class). In OOP, abstraction is a process of hiding the implementation details from the user, and only providing the functionality. In other words, the user will have the information on what the object does instead of how it does it.

In Java, the process of abstraction is done using interfaces, classes, abstract classes, fields, methods and variables. It is the fundamental concept on which other things rely on such as encapsulation, inheritance and polymorphism.

2.2 ENCAPSULATION

Encapsulation is one of the fundamental OOP concepts. It is a technique of wrapping (packaging) the related data (attributes) and behavior (code - methods) together into a single unit. It also provides a way to hide and control data by protecting it from misuse by the outside world. This is achieved by making data members private, and using public

helper methods through which we can decide the level of access we want to provide (e.g. read-only, write-only). A fully encapsulated Java class is a class whose all variables are private.

The advantages of encapsulation include: flexibility in modifying code in case requirements change, reusability of encapsulated code throughout multiple applications if necessary, and reducing the time of maintenance (update) process.

2.2.1 Visibility Modifiers

In Java, there are four access modifiers which provide various access levels: **private** (visible inside of the class only), **default/package** (visible inside of the package), **protected** (visible inside the package and to all subclasses) and **public** (visible to everyone).

2.3 INHERITANCE

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another class. It is achieved with the keyword `extends`. The class which inherits the properties of another class is known as subclass (derived class, child class), and the class whose properties are inherited is known as superclass (base class, parent class).

Inheritance enables managing information in a hierarchical order, code reuse, extending a class by adding new features instead of writing a new class from scratch, and maintainability of the code (e.g. updating the code in one place – superclass, instead of updating every single class).

2.4 POLYMORPHISM

Polymorphism is the ability of an object to take on many forms. In OOP, polymorphism means a type can point to different object at different time. In other words, the actual object to which a reference type refers, can be determined at runtime. Any Java object that can pass more than one IS-A test is considered to be polymorphic. Thus, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. A reference variable can refer to any object of its declared type or any subtype of its declared type.

Polymorphism increases code reuse, allows for flexibility in running code and code extensibility.

In Java, polymorphism is based on inheritance and overriding.

2.4.1 Method Overriding

When a class extends another class, it can use its super class's methods. However, sometimes the subclass may need a different behavior of a method provided by its superclass. The method implementation in the subclass overrides (replaces) the method implementation in the superclass. In this case, the subclass method and superclass method have the same name, parameters and return type, but different implementation. This is called method overriding.

2.5 METHOD OVERLOADING

Method overloading should not be confused with method overriding. When we need to have more than one method with the same functionality within the same class, we don't have to declare new methods with different names for each one. Method overloading feature allows us to declare multiple methods with the same name, but different signatures (e.g. different argument list, types or order). `System.out.println()` is an example of method overloading in Java. It takes float, int, double or String types as arguments.

2.6 UML CLASS DIAGRAMS

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system. A UML class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes,

their attributes, operations (or methods), and the relationships among objects. The class diagram is the main building block of object-oriented modelling.

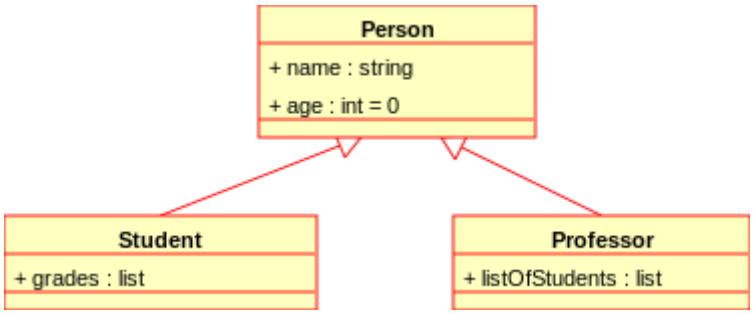
In a UML diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class in bold and centered, and the first letter is capitalized.
- The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

To specify the visibility of a class member (i.e. any attribute or method), these notations must be placed before the member's name:

+	Public
-	Private
#	Protected
/	Derived (can be combined with one of the others)
~	Package

A relationship is a general term covering the specific types of logical connections found on class and object diagrams. UML defines several relationships. In this assignment, you are expected to show only inheritance relationships (or "IS-A" relationship) between your classes. Example:



For this assignment, you are expected to draw a basic UML diagram for your classes, which will include class names, all attributes and methods with the specified visibility, and show inheritance relationships among the classes. You may use any tool for this task. For example, you can look up available UML generator tools for eclipse at https://marketplace.eclipse.org/.

2.7 JAVADOC

Javadoc is a documentation generator for Java, which Generates HTML pages of API documentation from Java source files. The basic structure of writing document comments is to embed them inside `/** ... */`. You need to submit Javadoc for your code as well. For details refer to:

http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

3 AN OVERVIEW OF THE APPLICATION

In this programming assignment, you will develop a very simplified version of an online shopping site. We will call it HUBBM Bazaar.

3.1 PROBLEM DEFINITION

First, we need to review the requirements (specifications) of the system we want to develop. The basic components of an online shopping application include:

- **users** (service providers and customers),
- **items for sale** (classified according to a certain specification, such as product type),
- **shopping-related concepts** (product pricing, sale campaigns, etc.) and **events/actions** (viewing product information, adding items to the shopping cart, viewing discounts, managing personal information, placing orders, etc.).

Your program will need to be able to handle simple requirements of an online shopping application. You will be given input files that contain information necessary for instantiating the users of the system, and items for sale. Furthermore, another input file containing commands for various transactions and actions will be provided, which you will use to test your application.

The necessary components of the system are explained in detail in the following sections.

3.1.1 Users

Our online shopping application will comprise three different types of users: Administrative and technical staff who are employees (service providers), and costumers who are clients. You are expected to design your user classes in a hierarchical structure. User information is as follows:

- Every **Person** in the system has a **name**, an **e-mail**, and a **date of birth**.
- Every **Employee** has a **salary**.
- An employee can either be an **Admin** or a **Technician**.
- An admin has a **password**.
- A technician can be **senior** or not.
- A **Customer** has a **customerID**, a **password**, a **balance** (money in their shopping account), **status** (which can be **CLASSIC**, **SILVER** or **GOLDEN**), **shopping cart**, and **order history**.
- **customerID** is assigned in order (starting from 1) to each added customer.
- Customer status is calculated as follows: every new customer is automatically assigned the CLASSIC status. A customer who spends at least 1000 $ is awarded the SILVER status. A customer who spends at least 5000 $ is awarded permanent GOLDEN status and counts as a VIP customer.
- **A customer with a SILVER status gets a discount rate of 10%** for all orders (even ones that include items with discount campaigns).
- **A customer with a GOLDEN status gets a discount rate of 15%** for all orders (even ones that include items with discount campaigns).
- Customer status limits and discount rates must be stored as constant static variables.

Users should be able to perform the following actions:

- **All users** should be able to:
  - Display personal data: name, e-mail, and date-of-birth.
- **Customers** should be able to:
  - Change their password own by providing the old password and a new one,
  - Update their balance (deposit money into their shopping account),
  - View active campaigns,
  - Add items to their shopping cart, if the item is available (if it exists and is in stock),
  - Clear their shopping cart,
  - Place orders for items in their shopping carts by providing their password,
  - Orders will be successful only if all of the following conditions are satisfied: the shopping cart is not empty, the customer has enough money in the shopping account, and the password provided matches the customer's password.
  - After each order, customer's status should be updated if necessary. Also, the stock amount of the bought items should be updated accordingly.
  - Customer class should also override `toString()` method to print the customer's data: ID, e-mail, date of birth and status.
- **All employees** should be able to:
  - Display stock amount and item types,
  - List all available items with their types,

    - Display VIP customers.
- **Admins** should be able to:
    - Add new customers to the system by providing the customer name, e-mail, date-of-birth, and initial customer balance and password,
    - Add other admins and technicians to the system,
    - Display all available data about a particular customer,
    - Display all customers,
    - Override the display personal data method to add the keyword "Admin" before their name, e-mail, and date-of-birth,
    - Launch a campaign.
- **Technicians** should be able to:
    - Display all orders made by all customers only if they are senior technicians,
    - Display info about a particular item,
    - Add a new item.

3.1.2 Items

Items represent all products that are available for sale. They are categorized as either **Cosmetic**, **Electronic**, or **Office Supplies** items. You are expected to design your item classes in a hierarchical structure. The necessary information about the items is as follows:

- Subcategories of Cosmetic items are: **Perfume**, **Hair care**, and **Skin care**.
- Subcategories of Electronic items are: **Computer**, **TV**, and **Smart phone**.
- Subcategories of Office Supplies items are: **Book**, and **CD-DVD**.
- Subcategories of Computer items are: **Desktop**, **Laptop**, and **Tablet**.
- All items have an **ID** and a **price**.
- **Cosmetic** and **Electronic** items possess **manufacturer** and **brand** information.
- **ID** is assigned in order (starting from 1) to each added item.
- Cosmetic items have an **expiration date**, a **weight**, and can be either **organic** or not.
- Perfumes have **lasting duration**.
- Hair care items may be **medicated** or not.
- Skin care items may be **baby sensitive** or not.
- Electronic items have a **maximum allowed input voltage** and a **maximum power consumption** in Watts.
- Computer items have an **operating system**, a **CPU Type**, a **RAM Capacity**, and an **HDD Capacity** in gigabytes.
- Desktop computer items have a **box color**.
- Laptop computer items may have **HDMI support** or not.
- Tablet computer items have **dimensions**.
- TVs have a **screen size -**in inches.
- Smart phone items have a **screen type**.
- Office Supplies items have a **release date**, and a **cover title**.
- Book items have a **publisher**, **author(s)**, and a number of **pages**.
- CD-DVD items have a **composer**, and **song(s)**.
- Apart from the above, all item types (Laptop, Book etc.) have a related stock (inventory) information whose values should be initially obtained from the input file named `item.txt`.

3.1.3 Shopping

The main shopping concepts in our application are shopping **Orders** and discount **Campaigns**.

- Orders have an **orderDate**, a **totalCost**, list of **purchased items**, and **customerID** of the buyer.
- When order data is displayed; order date, customerID, total cost, and a number of bought items should be shown.
- Campaigns have a **startDate**, an **endDate**, **itemType**, and a **discountRate**.
- The maximum discount rate of any campaign is 50%.

3.1.4 Events/Actions

i. **Add a new customer**: A service provider with admin role is the only one authorized to add a new customer. The program should add a new customer if the syntax of the command is as provided below:

ADDCUSTOMER<TAB>**adminName**<TAB>**customerName**<TAB>**customerMail**<TAB>**customerDateofBirth**<TAB>**initialBalance**<TAB>**password**

Note: The first argument indicates the name of the admin who initiates the request and the program you implement has to check the records before perform this request. If no record is found with **adminName** then it should display an error message. For such cases or details, read *Constraints* section carefully.

Usage Illustration:

ADDCUSTOMER<TAB>**Cemile**<TAB>**Kerem**<TAB>**kerem@yahoo.com**<TAB>**21.02.1993**<TAB>**100000**<TAB>**kerem1111**
No admin person named Cemile exists!

ii. **Show customer information**: An admin person can also show the customers which are recorded so far. The displayed customer records should include: name, ID, e-mail, date of birth, and status. The syntax to display customer info should be as follows:

SHOWCUSTOMER<TAB>**adminName**<TAB>**customerID**

Usage Illustration:

	SHOWCUSTOMER<TAB>**Leyla**<TAB>**3**
Possible outputs	No admin person named Leyla exists! *(or)*
	Customer name: Hamza ID: 3 e-mail: hamza@hacettepe.edu Date of Birth: Tue Sep 08 00:00:00 EEST 1987 Status: CLASSIC

iii. **List customers**: Similar to the previous event, an admin is also authorized to list all customer info for customers who are recorded in the system. To do that the command syntax should be:

SHOWCUSTOMERS<TAB>**adminName**

Usage Illustration:

	SHOWCUSTOMER<TAB>**Demet**
Possible outputs	No admin person named **Demet** exists!
	Customer name: Emre ID: 1 e-mail: emre@hacettepe.edu Date of Birth: Mon Oct 02 00:00:00 EEST 2000 Status: CLASSIC Customer name: Meltem ID: 2 e-mail: meltem@hacettepe.edu Date of Birth: Fri Mar 30 00:00:00 EEST 1990 Status: CLASSIC

iv. **Show admin information**: Admin staff can also display their information. Note that they cannot display others' personal information.

SHOWADMININFO<TAB>**adminName**

Usage Illustration:

	SHOWADMININFO<TAB>**Can**
Possible outputs	No admin person named Leyla exists! *(or)*
	----------- Admin info ----------- Admin name: Can Admin e-mail: can@hacettepe.edu Admin date of birth: 21.05.1980

v. **Launch a campaign**: In the system, only admins are allowed to launch a new campaign.

CREATECAMPAIGN<TAB>**adminName**<TAB>**startDate**<TAB>**endDate**<TAB>**itemType**<TAB>**rate**

Note: There are a couple of constraints for launching a campaign. Please refer to Constraints section.

Usage Illustration:

	CREATECAMPAIGN<TAB>**Alper**<TAB>**23.03.2017**<TAB>**01.06.2017**<TAB>**BOOK**<TAB>**25**
Possible outputs	No admin person named Alper exists! *(or)*
	Campaign was not created. Discount rate exceeds maximum rate of 50%.

vi. **Add a new employee**: An existing user with admin role can add a new employee (another admin or a technician). To do that the command should be provided to the system as given below:

ADDADMIN<TAB>**adminName**<TAB>**newAdminName**<TAB>**newAdminMail**<TAB>**newAdminDateofBirth**<TAB>**newAdminSalary**<TAB>**newAdminPassword**

ADDTECH<TAB>**adminName**<TAB>**newTechName**<TAB>**newTechMail**<TAB>**newTechDateofBirth**<TAB>**newTechSalary**<TAB>**isSenior**

Usage Illustration:

	ADDADMIN<TAB> **Demet**<TAB> **Yavuz**<TAB> **yavuz@gmail.com**<TAB>**16.07.1984**<TAB> **1000**<TAB> **yavuz1** **ADDTECH**<TAB>**Yavuz**<TAB> **Kubilay**<TAB> **kubilay@outlook.com**<TAB>**19.04.2001**<TAB> **500**<TAB>**1**
Possible outputs	No admin person named Demet exists! *(or)*
	No technician person named Yavuz exists!

vii. **List existing items**: Service providers – or employees (admins and technicians) can list the existing items with their information in detail.

LISTITEM<TAB>**[adminName|technicianName]**

Usage Illustration:

	LISTITEM<TAB>Kubilay //assume Kubilay is an employee (admin or tech.)
Possible outputs	No admin or technician person named Kubilay exists! *(or)*
	Type: SmartPhone Item ID: 28 Price: 1100.0 $ Manufacturer: Apple Inc. Brand: APPLE Max Volt: 200 V. Max Watt: 240 W. Screen Type: IPS LCD ---------- Type: TV Item ID: 25

	Price: 12000.0 $ Manufacturer: Royal Philips Electronics of the Netherlands Brand: PHILIPS Max Volt: 200 V. Max Watt: 240 W. Screen size: 74" . . .

viii. **Show items types with low stock**: In this system, one of the missions of service providers (admins or technicians) is to monitor the stock status of every item type. Thus, they should often display the remaining stock amount of items. The system response to a stock status request differs depending on the syntax. In the first case, the system considers the provided stock value and displays only those items whose stock amount is below this limit value if provided in the command line; otherwise, the system displays stock information of all items.

SHOWITEMSLOWONSTOCK<TAB>[**adminName|technicianName]**<TAB>**{maxStockAmount}**

Usage Illustration:

	SHOWITEMSLOWONSTOCK<TAB>**Enes**<TAB>**14** **SHOWITEMSLOWONSTOCK**<TAB>**Enes**
Possible outputs	No admin or technician person named Enes exists! *(or)*
	Book : 16 CDDVD : 5 Desktop : 11 Laptop : 42 Tablet : 11 TV : 13 SmartPhone : 9 HairCare : 4 Perfume : 1 SkinCare : 8
	CDDVD : 5 Desktop : 11 Tablet : 11 TV : 13 SmartPhone : 9 HairCare : 4 Perfume : 1 SkinCare : 8

ix. **Show VIP customers**: Service providers can display VIP customers. VIP customers are those whose status is *GOLDEN*.

SHOWVIP<TAB>[**adminName|technicianName]**

Usage Illustration:

	SHOWVIP<TAB>**Alper**
Possible outputs	No admin or technician person named Alper exists!
	Customer name: Emre ID: 1 e-mail: emre@hacettepe.edu Date of Birth: Mon Oct 02 00:00:00 EEST 2000 Status: GOLDEN

x. **Display items under specific types**: A technician can display sub-items. To do that he/she should provide item types each of which is separated by a colon mark.

DISPITEMSOF<TAB>**technicianName**<TAB>**type1**<:>**type2**<:>...<:>**typeN**

Usage Illustration:

	DISPITEMSOF<TAB>**Kubilay**<TAB>**BOOK:HAIRCARE**
Possible outputs	No technician person named Kubilay exists! *(or)*
	Type: Book Item ID: 4 Price: 4.0 $ Release Date: 1960 Title: To Kill a Mockingbird Publisher: J. B. Lippincott & Co. Author: Harper Lee Page: 281 pages ---------- Type: HairCare Item ID: 30 Price: 26.0 $ Manufacturer: Goldwell Manufacturing Services Brand: GOLDWELL Organic: Yes Expiration Date: 2021 Weight: 750 g. Medicated: Yes

xi. **Add a new item**: Only technicians are authorized to add new items to the system. To add a new item, they should take argument length and order of the item (See Items Input File Section) into consideration. Colon marks separate the arguments.

ADDITEM<TAB>**technicianName**<TAB>**itemType**<:>**argument1**<:>**argument2**<:>...<:>**argumentN**

Usage Illustration:

	ADDITEM<TAB>**Kubilay**<TAB> **LAPTOP:1250:Dell Inc.:DELL:220:250:Windows 10 Home:Intel Core i7:8:250:1**
Possible outputs	No technician person named Kubilay exists!
	No item type Laptop found

xii. **Show orders**: Another and the last mission of the tech. team is to display the orders requested so far. As mentioned, not all technicians can visualize orders, but only the ones who are senior technicians.

SHOWORDERS<TAB>**technicianName**

Usage Illustration:

	SHOWORDERS<TAB>**Emir**
Possible outputs	No technician person named Emir exists!
	Emir is not authorized to display orders!
	Order History:

	Order date: Mon Mar 20 12:18:39 EET 2017 Customer ID: 1 Total Cost: 12000.0 Number of purchased items: 0

xiii. **Change password**: Customers use their passwords when placing a new order request, which is designed to avoid transactions on behalf of someone else. Passwords are initially determined by admins who are authorized to add new customers, then customers can change their passwords using this command:

CHPASS<TAB>**customerID**<TAB>**oldPassword**<TAB>**newPassword**

Usage Illustration:

	CHPASS<TAB>**1**<TAB>**emre1234**<TAB>**1234emre**
Possible outputs	No customer with ID number 1 exists!
	The given password does not match the current password. Please try again.
	The password has been successfully changed.

xiv. **Load money**: Customers can place a new order only if their balances are sufficient. Before placing a new order they can deposit money into their accounts if it does not have sufficient amount for purchase.

DEPOSITMONEY<TAB>**customerID**<TAB>**loadAmount**

Usage Illustration:

	DEPOSITMONEY<TAB>**customerID**<TAB>**loadAmount**
Possible outputs	No customer with ID number 1 exists!

xv. **Display campaigns**: Customers may prefer to buy items that have a sale campaign. To display campaigns, the command will be as given below:

SHOWCAMPAIGNS<TAB>**customerID**

Usage Illustration:

	SHOWCAMPAIGNS<TAB>**1**
Possible outputs	No customer with ID number 1 exists!
	No campaign has been created so far!
	Active campaigns: 20% sale of PERFUME until 01/09/2017

xvi. **Add to cart**: To enable a customer to buy more than one item within one order, almost all online shopping system provide a cart mechanism. This system also employs this mechanism, and customers should first add some items to their carts before placing a purchase order. To do that, the command line below should be provided to the system:

ADDTOCART<TAB>**customerID**<TAB>**itemID**

Usage Illustration:

	ADDTOCART<TAB>**1**<TAB>**11**
Possible outputs	No customer with ID number 1 exists!
	Invalid item ID
	We are sorry. The item is temporarily unavailable.
	The item Book has been successfully added to your cart.

xvii. **Empty cart**: In a shopping system, customers may want to empty their carts and start shopping from scratch. The command syntax below enables customer to do that:

EMPTYCART<TAB>**customerID**

Usage Illustration:

	EMPTYCART<TAB>**1**
Possible outputs	No customer with ID number 1 exists!
	The cart has been emptied.

xviii. **Place a new purchase order**: The process of shopping activity of customers ends with a purchase order if they want to buy the items in their shopping carts. To direct customer to the payment phase, the system should first consider the customer's password and check if it matches the password of the customer whose ID is given. If it does, the system should calculate the total price of the items in the customers' carts considering customers' status (*GOLDEN*, *SILVER*, or *CLASSIC*) and active campaigns. As mentioned, if the status of the customer is other than *CLASSIC* then the system should lower the total price depending on their status. Similarly, if a campaign has already been launched for an item that is in a customer's cart, the system should also lower the price only for that item. To successfully place an order, the customer should have enough money, which means the system should also check the balance of the customer who initiated the order process. After a successful order operation, the system should give a feedback to the customer informing him/her about how much more money they need to spend until their customer status can be upgraded (to SILVER or GOLDEN when applicable).

ORDER<TAB>**customerID**<TAB>**customerPassword**

Usage Illustration:

	ORDER<TAB>**1**<TAB>**emre1234**
Possible outputs	No customer with ID number 1 exists!
	The cart has been emptied.
	Order could not be placed. Insufficient funds.
	You should add some items to your cart before order request!
	Order could not be placed. Invalid password.
	Done! Your order will be delivered as soon as possible. Thank you!
	Congratulations! You have been upgraded to a XXX MEMBER! You have earned a discount of X% on all purchases.
	You need to spend XXX more TL to become a XXX MEMBER.

4 INPUT FILES

You will be provided with three (tab-separated) text input files:

- **Users** in the system,
- **Items** that are sold,
- **Commands** that you need to execute.

4.1 USERS INPUT FILE

List of all users in the system (admins, technicians and customers) will be given in the (tab-separated) `users.txt` file:

- a line containing information about an admin will start with a keyword **`ADMIN`**,
- a line containing information about a technician will start with a keyword **`TECH`**,
- a line containing information about a customer will start with a keyword **`CUSTOMER`**.

The format of each person's data will be given as follows:

`ADMIN`<TAB>**`name`**<TAB>**`email`**<TAB>**`birthDate`**<TAB>**`salary`**<TAB>**`password`**

`TECH`<TAB>**`name`**<TAB>**`email`**<TAB>**`birthDate`**<TAB>**`salary`**<TAB>**`isSeniorTechnician`**

`CUSTOMER`<TAB>**`name`**<TAB>**`email`**<TAB>**`birthDate`**<TAB>**`balance`**<TAB>**`password`**

A sample `users.txt` input file is given below:

```
ADMIN  Demet  demet@hacettepe.edu 11.12.1989   3500   demet1234
CUSTOMER      Emre   emre@hacettepe.edu  02.10.2000   0      emre1234
TECH   Fatih  fatih@hacettepe.edu 17.03.1992   2100   0
CUSTOMER      Meltem meltem@hacettepe.edu       30.03.1990   500.40 meltem1234
CUSTOMER      Hamza  hamza@hacettepe.edu 08.09.1987   10321.5      hamza1234
ADMIN  Alper  alper@hacettepe.edu 19.12.1991   3500   alper1234
TECH   Emir   emir@hacettepe.edu  28.02.1983   2700   1
ADMIN  Can    can@hacettepe.edu   21.05.1980   3450   can1234
ADMIN  Leyla  leyla@hacettepe.edu 01.11.1975   3600   leyla1234
ADMIN  Cemil  cemil@hacettepe.edu 06.07.1985   3750   cemil1234
TECH   Handan handan@hacettepe.edu       29.10.1989   2700   1
CUSTOMER      Taha   taha@hacettepe.edu  29.04.1969   7505.43      taha1234
CUSTOMER      Furkan furkan@hacettepe.edu       30.09.1974   153.85 furkan1234
TECH   Enes   enes@hacettepe.edu  02.02.1996   2100   0
```

You are expected to create an instance for each user in this file.

4.2 ITEMS INPUT FILE

List of all items for sale will be given in the (tab-separated) `item.txt` file.

- a line containing information about a desktop computer item will start with a keyword **`DESKTOP`**,
- a line containing information about a notebook computer item will start with a keyword **`LAPTOP`**,
- a line containing information about a tablet item will start with a keyword **`TABLET`**,
- a line containing information about a television item will start with a keyword **`TV`**,
- a line containing information about a smart phone item will start with a keyword **`SMARTPHONE`**,
- a line containing information about a book item will start with a keyword **`BOOK`**,
- a line containing information about a CD-DVD item will start with a keyword **`CDDVD`**,
- a line containing information about a hair care item will start with a keyword **`HAIRCARE`**,
- a line containing information about a skin care item will start with a keyword **`SKINCARE`**,
- a line containing information about a perfume item will start with a keyword **`PERFUME`**.

The format of each item data will be given as follows:

`DESKTOP`<TAB>**`cost`**<TAB>**`manufacturer`**<TAB>**`brand`**<TAB>**`maxVolt`**<TAB>**`maxWatt`**<TAB>**`operatingSystem`**
<TAB>**`CPUType`**<TAB>**`ramCapacity`**<TAB>**`HDDCapacity`**<TAB>**`boxColor`**

`LAPTOP`<TAB>**`cost`**<TAB>**`manufacturer`**<TAB>**`brand`**<TAB>**`maxVolt`**<TAB>**`maxWatt`**<TAB>**`operatingSystem`**
<TAB>**`CPUType`**<TAB>**`ramCapacity`**<TAB>**`HDDCapacity`**<TAB>**`HDMISupport`**

TABLET<TAB>**cost**<TAB>**manufacturer**<TAB>**brand**<TAB>**maxVolt**<TAB>**maxWatt**<TAB>**operatingSystem**
<TAB>**CPUType**<TAB>**ramCapacity**<TAB>**HDDCapacity**<TAB>**dimension**

TV<TAB>**cost**<TAB>**manufacturer**<TAB>**brand**<TAB>**maxVolt**<TAB>**maxWatt**<TAB>**screenSize**

SMARTPHONE<TAB>**cost**<TAB>**manufacturer**<TAB>**brand**<TAB>**maxVolt**<TAB>**maxWatt**<TAB>**screenType**

BOOK<TAB>**cost**<TAB>**releaseDate**<TAB>**coverTitle**<TAB>**publisherName**<TAB>**author1**<,>
author2<,>...<,>**authorN**<TAB>**pageNumber**

CDDVD<TAB>**cost**<TAB>**releaseDate**<TAB>**coverTitle**<TAB>**composerName**<TAB>**song1**<,>**song2**<,>...
<,>**songN**

HAIRCARE<TAB>**cost**<TAB>**manufacturer**<TAB>**brand**<TAB>**isOrganic**<TAB>**expirationYear**<TAB>
weight<TAB>**isMedicated**

SKINCARE<TAB>**cost**<TAB>**manufacturer**<TAB>**brand**<TAB>**isOrganic**<TAB>**expirationYear**<TAB>
weight<TAB>**babySensitive**

PERFUME<TAB>**cost**<TAB>**manufacturer**<TAB>**brand**<TAB>**isOrganic**<TAB>**expirationYear**<TAB>
weight<TAB>**lastingDuration**

A sample `item.txt` input file is given below:

BOOK	2	2016	Everything We Keep	Lake Union	Kerry Lonsdale	306						
BOOK	25	1992	Ulysses	Modern Library	James Joyce 844							
BOOK	3	2006	Nick and Norah's Infinite Playlist	Alfred Knopf Books	Rachel Cohn,David Levithan	183						
BOOK	4	1960	To Kill a Mockingbird	J. B. Lippincott & Co.	Harper Lee	281						
BOOK	7	2004	Sorcery and Cecelia or The Enchanted Chocolate Pot	HMH Books	Patricia C. Wrede,Caroline Steverme,Ayn Rand	336						
CDDVD	10	2016	Live North America	Gary Clark Jr.	Grinder,Our Love,When My Train Pulls In,Church							
CDDVD	5	2008	One of the Boys	Katy Perry	One Of The Boys,I Kissed A Girl,Thinking Of You,If You Can Afford Me							
CDDVD	13	2012	Red	Taylor Swift	State Of Grace,Red,I Almost Do							
CDDVD	21	1959	Kind of Blue	Miles Davis	So What,Freddy Freeloader							
CDDVD	18	1957	The Great Ray Charles	Ray Charles	The Ray,The Man I Love,Hornful Soul							
DESKTOP	1250	Micro-Star International	MSI	220	250	Free-Dos	Intel Core i5	8	750	black		
DESKTOP	1430	AsusTek Computer Inc.	ASUS	220	250	Windows 10 Home	Intel Core i7	16	1000	white		
DESKTOP	1000	AsusTek Computer Inc.	ASUS	220	250	Free-Dos	Intel Core i3	8	500	red		
DESKTOP	2100	Dell Inc.	DELL	220	250	Windows 10 Home	Intel Core i7	16	2000	black		
DESKTOP	2400	Apple Inc.	APPLE	220	250	MAC OS X Yosemite	Intel Core i5	8	500	white		
LAPTOP	3100	AsusTek Computer Inc.	ASUS	220	250	Windows 10 Home	Intel Core i7	16	750	1		
LAPTOP	3400	Apple Inc.	APPLE	220	250	MAC OS	Intel Core i5	8	1000	0		
LAPTOP	1800	Hewlett-Packard Company		HP	220	250	Free-Dos	Intel Core i7	16	300	1	
LAPTOP	1700	AsusTek Computer Inc.	ASUS	220	250	Windows 10 Home	Intel Core i3	8	500	0		
LAPTOP	2050	Dell Inc.	DELL	220	250	Free-Dos	Intel Core i7	16	400	1		
TABLET	90	AsusTek Computer Inc.	ASUS	220	100	Android 4.4 (KitKat)	Qualcomm Quad-core	1	8	9		
TABLET	135	Samsung Electronics	SAMSUNG	220	100	Android 5.0.2 (Lollipop)	Samsung Exynos	3	32	11		
TV	900	Royal Philips Electronics of the Netherlands		PHILIPS	200	240	43					
TV	1500	Samsung Electronics	SAMSUNG	200	240	40						
TV	12000	Royal Philips Electronics of the Netherlands		PHILIPS	200	240	74					
SMARTPHONE		700	Samsung Electronics	SAMSUNG	200	240	Quad HD Super AMOLED					
SMARTPHONE		600	Sony Electronics Manufacturing		SONY	200	240	Super AMOLED				
SMARTPHONE		1100	Apple Inc.	APPLE	200	240	IPS LCD					
HAIRCARE	22	Henkel AG & Company	SCHWARZKOPH		1	2019	1000	0				
HAIRCARE	26	Goldwell Manufacturing Services		GOLDWELL	1	2021	750	1				
PERFUME	13	Calvin Klein Inc.	CALVIN KLEIN		0	2025	250	75				
PERFUME	17	Hugo Boss Group	HUGO BOSS		0	2021	330	110				
PERFUME	11	Calvin Klein Inc.	CALVIN KLEIN		0	2019	500	30				
SKINCARE	19	Beiersdorf Global AG.	NIVEA	1	2017	150	1					
SKINCARE	13	Clinique Laboratories	CLINIQUE	0	2025	750	1					
SKINCARE	9	Beiersdorf Global AG.	NIVEA	0	2022	400	0					

You are expected to create an instance for each item in this file.

4.3 COMMANDS INPUT FILE

List of all commands, which will be given to test your program and which you are expected to execute correctly and in order, will be given in the (tab-separated) `commands.txt` file. A detailed explanation, format and expected output for each command is given in the Events/Actions section (Section 3.1.4) of this document.

A sample `commands.txt` input file is given below:

```
ADDCUSTOMER Cemil Kerem kerem@yahoo.com   21.02.1993  100000      kerem1111
SHOWCUSTOMER      Leyla 3
SHOWCUSTOMERS     Demet
SHOWCUSTOMERS     Ferit
ADDCUSTOMER Musa  Ayten ayten@yahoo.com   05.10.1981  1000  ayten0000
ADDTOCART   7     37
ORDER 7     ayten0000
SHOWCUSTOMER      Can   4
SHOWADMININFO     Can
SHOWADMININFO     Enes
CREATECAMPAIGN    Alper 23.03.2017  01.06.2017  BOOK  25
CREATECAMPAIGN    Leyla 21.03.2017  01.09.2017  DEKSTOP     90
CREATECAMPAIGN    Leyla 21.03.2017  01.09.2017  PERFUME     20
SHOWCAMPAIGNS     2
ADDTOCART   3     10
ADDTOCART   3     3
ADDTOCART   3     5
ADDTOCART   3     15
ADDTOCART   5     10
ORDER 3     hamza1234
DEPOSITMONEY      6     210.6
CHPASS      1     emre1234    emre12345678
EMPTYCART   3
ORDER 3     hamza1234
ADDTOCART   1     15
ADDTOCART   1     2
ADDTOCART   1     11
ORDER 1     emre12
ORDER 1     emre1234
ORDER 1     emre12345678
SHOWORDERS  Emir
SHOWITEMSLOWONSTOCK     Enes  14
SHOWVIP     Alper
ADDADMIN    Demet Yavuz yavuz@gmail.com   16.07.1984  1000  yavuz1
SHOWADMINONFO     Yavuz
ADDTECH     Yavuz Kubilay     kubilay@outlook.com     19.04.2001  500   1
ADDITEM     Kubilay     LAPTOP:1250:Dell   Inc.:DELL:220:250:Windows   10   Home:Intel   Core
i7:8:250:1
ADDTOCART   6     37
ADDTOCART   6     25
ORDER 6     kerem1111
SHOWORDERS  Emir
SHOWCUSTOMERS     Demet
LISTITEM    Kubilay
DISPITEMSOF Kubilay     BOOK:HAIRCARE:PERFUME
```

5 JAVADOC

Javadoc is the JDK tool that generates API documentation from documentation comments. You are expected to document every method and class using Javadoc. Documentation comments (doc comments) are special comments in the Java source code that are delimited by the /** ... */ delimiters. These comments are processed by the Javadoc tool to generate the API docs.

The Javadoc tool can generate output originating from four different types of "source" files. In this assignment, the source code files we are interested in are your Java classes (.java) - these contain class, interface, field, constructor and method comments.

A format of a doc comment is as follows:

- A doc comment is written in HTML and must precede a class, field, constructor or method declaration. It is made up of two parts -- a description followed by block tags. In this example, the block tags are `@param`, `@return`, and `@see`.

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
       try {
           return getImage(new URL(url, name));
       } catch (MalformedURLException e) {
           return null;
       }
}
```

- The first sentence of each doc comment should be a summary sentence, containing a concise but complete description of the API item. This means the first sentence of each member, class, interface or package description. The Javadoc tool copies this first sentence to the appropriate member, class/interface or package summary. This makes it important to write informative initial sentences that can stand on their own. E.g.

```
/**
    * Class constructor.
    */
   foo() {
     ...

   /**
    * Class constructor specifying number of objects to create.
    */
   foo(int n) {
     ...
```

Tag conventions:

- Order of tags should be:
  - `@author` (classes and interfaces only, required)
  - `@version` (classes and interfaces only, required)
  - `@param` (methods and constructors only)
  - `@return` (methods only)
  - `@exception` (@throws is a synonym added in Javadoc 1.2)
  - `@see`
- Required tags:
  - An `@param` tag is "required" (by convention) for every parameter, even when the description is obvious.
  - The `@return` tag is required for every method that returns something other than `void`, even if it is redundant with the method description. (Whenever possible, find something non-redundant (ideally, more specific) to use for the tag comment.)

For more information on Javadoc and examples of doc comments, visit:
http://www.oracle.com/technetwork/articles/java/index-137868.html

Important note: DO NOT use Turkish characters in your code (anywhere, not even comments)!

6 CONSTRAINTS

- **Person Existence**: As you may have realized, every command has its operator who may be an admin, a technician, or a customer. The command's operator name or ID (for customers) is provided as the second argument of the command line. The system should output an error message if such person does not exist in the system.
- **Stock Update**: As new items are added to the system or ordered by the customers, stock amount of the item type to which item belongs should be updated accordingly.
- **Add New Item**: The arguments of an item to be added to the system should be separated by colon marks (:). Before processing an **Add New Item** request, the system should first check if the item type has already been defined before.
- **Change Password**: Customers can update their password, but the system should check if the provided password matches the current password.
- **Show Campaigns**: The system allows a customer to visualize existing campaigns. However, there might be no campaigns launched so far. In such case, the system should output a message whose content is given in the Action/Event section.
- **Launch New Campaign**: To launch a new campaign successfully, the discount rate should not exceed the maximum rate allowed in the system. An admin launches a campaign not for a particular item but for all items under a specified type. If a campaign already exists for that type, the system should not allow the admin to create another one.
- **Show Orders**: To visualize the orders made so far, system should consider the senior status of technician who initiates display request.
- **Add to Cart**: In this request, the system should check if there actually is such item whose ID number is provided by a customer. If no item with *itemID* exists in the system, an error message should be displayed (the content of this message is also provided in Action/Event section). Moreover, the system should also take stock information of the item into consideration. If the stock is depleted, the system should prevent customers from adding that item to their carts.
- **Make Order**: When processing this request, the system should first consider the password of the customer who initiates the order request. The system should also check if the customer's cart is empty. If everything goes well up to this stage, the system should now calculate the total price of the order. The customer should have enough money to purchase all items in his/her cart. The system should also check the active campaigns for each item in the customer's cart, and if a campaign exists, the price for that item should be lowered depending on the discount rate defined in the campaign. For the total price calculation, the system should also consider the customer's present status. If his/her status is other than CLASSIC, the price should also be lowered by the system. After the process of placing order is over, the system should update the stock amount of each purchased item, update the customer's status if necessary, and calculate and display the amount of money that the customer should spend in order to pass to the higher status (if applicable).
- **General constraints**:
  i. All outputs should be printed to the console and should match output formats provided in the Section Events/Actions.
  ii. Use comments as much as possible throughout your code, even for each method in a class.
  iii. Do not use Turkish characters (**not even in your comments**!) as they causes compilation errors.
  iv. Do not submit your project without first compiling it on dev machine.
  v. Make sure your implementation works without runtime or compile errors.
  vi. Ignore the cases which are not stated in this assignment and do not ask questions on Piazza for such extreme cases.
  vii. In general, if something is not specifically forbidden by this document, you may assume that it is allowed.

7 SUBMISSION

- Submissions will be accepted only electronically via submit.cs.hacettepe.edu.tr.
- The deadline for submission is **12.04.2017 until 23:59 (late submissions will not be accepted!)**.
- The **minimum** submission format is (minimum meaning that your zipped folders may contain other files/folders):

  b<student id>.zip
  - javadoc
  - src
  - uml.jpg

- **Your main class must be named Main.java**! All of your classes should be in the **src** folder (they may, however, be further classified into other subfolders in case you decide to make packages). You may zip your whole eclipse project folder, but don't forget to include your drawing of UML class diagrams saved as `uml.jpg`. Note that only zipped folders can be submitted.
- **All work must be individual**! Plagiarism check will be performed! Duplicate submissions will be graded with 0 and disciplinary action will be taken.

8 COMPILE & RUN

javac Main.java

java Main users.txt item.txt commands.txt