**Objectives**

This assignments aims to:

1. Train students on Java and its IDEs like Eclipse, JCreator and NetBeans.
2. Train students on basic software engineering practices (code style and code commenting).

**Setup**

1. This assignment will be solved in **groups of 3 students** **from the same lab**.
2. Each team member will pick one problem from task 1 and one option from task 2. All the team will collaborate on task 3.
3. The group will submit together a combined solution.
4. 4 marks go to individual work and 2 marks go to collaboration.
5. There is 1 bonus mark given individually for outstanding achievement.
6. The group should select a leader to be responsible of following up their work.
7. The entire group is responsible of helping any weak member to be able to do his/her task **by his/herself**, by providing the necessary support, knowledge, hands-on demos, etc.
8. TA can ask any group member about the work on any other group member.
9. مسؤولية الفريق تضامنية عن عمله و أى غش من أى فرد سيكون مسؤولية الجميع و يخصم منهم 3 أضعاف الدرجة
10. Lab 1 document is included. **It is very important you do Lab 1 to master Java OOP concepts.**
11. **Please read the marking criterion very carefully to understand how you will be marked.**

**Task 1 (2%)**

In this task, each team member will pick one of the following problems and solve it. **Team should ensure that they agree on the public methods of each class.** So, when they integrate their work (put the classes together), it works well. All team members must:

* **Apply OOP principles** properly.
* **Use the coding style guide attached to this file.**
* **Write Java Doc comments** for your code **and generate Java Doc pages.**

Problem 1

We are going to implement a bank system. It's very important to maintain data of each account in the bank. So, we need to create an **Account** class. We will also create a **SpecialAccount** class that inherits from **Account** class and has a special feature.

* Create **Account** class.
* Create **balance** and **account number** attributes in account class (private not public)
* Create a suitable constructor that sets the **account number** and the initial **balance** value.
* Create setter and getter for each attribute.
* Override the method **toString ( )** inherited from class Object to make it return a meaningful string representation of the account information.
* Create methods **withdraw** and **deposit** to be able to take or put money. To withdraw, enough balance should be available.
* Create another class **SpecialAccount** that inherits from class **Account**.
* Use polymorphism to override method **withdraw** to allow over drafting with maximum limit of 1000 LE. اوفر درافت معناها السحب على المكشوف يعنى ممكن يسحب فلوس من الحساب حتى لو رصيده صفر بحد أقصى 1000 جنيه يعنى ممكن يصل رصيد الحساب إلى -1000 جنيه
* Write a main class to test class **Account**. It should create two accounts of two types and test all the functions you created in them.

Problem 2

We are going to implement a bank system. It is required to implement a **Client** class to represent the bank’s clients. So, you are required to do the following:

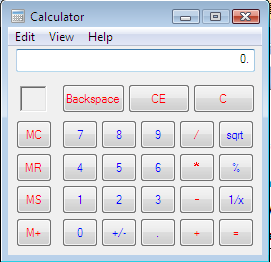
* Create **Client** class.
* Add **name, nationalID, address, phone** and **account** private attributes for the class.
* Create a suitable constructor that sets these attributes. (Note that before creating a new **Client,** you will need to create an object of type **Account** and pass it as a parameter to the client object’s constructor to be his account).
* Create setter and getter for each attribute.
* Override the method **toString ( )** inherited from class Object to make it return a meaningful string representation of the client information. String representing Client information, should also include his account’s information.
* Create another class **CommercialClient** that inherits from class **Client**. A commercial client is a company not a person. It has an extra attribute **commercialID** and setter and getter for it. Its **NationalID** is set to 00000000000000.
* Use polymorphism to override method **toString** print the commercial client details including his commercial ID instead of the national ID.
* Write a main class to create two clients of two types and test all the functions you created in them.

Problem 3

Now you have Account and Client classes and their subclasses (assume they exist until your colleagues finish doing the real ones). Let's make bank class

* Create **Bank** class.
* Create **name**, **address** and **phone** attributes in **Bank** class.
* Create a constructor and suitable setters and getters.
* Create an **ArrayList** of accounts in **Bank** class, this array maintains all accounts’ data.
* Create an **ArrayList** of clients in **Bank** class, this array maintains all clients’ data.
* Create a method to display a menu of options that allows the user to (1) add a new client and his account (can be special client or commercial account also) and (2) display existing clients and their accounts. Each time a new client and account are created, they are added to the two **ArrayLists** of accounts and clients.
* **Integrate your Bank class with Client and Account classes and use them in it.**
* Test your banking system in the **main** method**.**

**Task 2 (2%)**



In this task, each team member will pick one of the following software ideas, design it, implement it, test it, run it and answer a list of questions about it. **In your program, (1) apply as many OOP concepts as you can. See Lab 1 for all concepts that you need to know. (2) Also apply the coding style provided to your program so it is nicely organized and readable. (3) Finally, use Javadoc tags to document your code. And generate Html documentation from it.**

**Idea 1**

It is required to develop a small simple calculator

similar to Window's standard calculator but with

simpler functionality, e.g., no memory or storage

of results. It just performs an operation, stores the results, then can perform another operation using the last result, etc. and finally display the final result. Or, it can clear all the results to start another operation. **Do not write one big fat class. Use OOP concepts to write a nicely designed and understandable program that is broken reasonably into a number of classes.**

**Idea 2**

Develop a class hierarchy whose root is an ***abstract*** class **Appliance** أجهزة منزليةthat has the general features of any appliance in an appliance store (e.g. make, mode, price, warranty, etc.). Then develop children classes representing **WashingMachine**, **Refrigerator** and **Stove**. Add at least 3 attributes and 3 operations to these appliances (other than constructor and setters and getters).

Then write a small Java program to simulate the appliance store that allows the owner to add new items to the appliances’ stock and to sell existing items. It is also possible to view or list the entire stock. Use polymorphism to be able to store the whole stock in an array of **Appliance**. Override **toString** method to be able to return a string representing the details of an appliance.

Assume any necessary other details. **Use as many OOP concepts as you can.**

**Idea 3**

Create a class representing the general characteristics of a two-players **NimGame** <http://en.wikipedia.org/wiki/Nim>. This class should have the necessary methods to allow two players to exchange turns and make choices, and track the result. The details of the game rules will be defined in sub-classes.

Then inherit from it a special type called **The 100 Game**  [http://en.wikipedia.org/wiki/Nim#The\_100\_game](http://en.wikipedia.org/wiki/Nim" \l "The_100_game).Then write a program to instantiate the class and play the game. **Create any necessary other classes.**

**Idea 4**

**نمذجة اللحوم الأسبانية**

الجزء الأول من التمرين يشمل نمذجة شرائح الشواء من اللحوم الأسبانية و تنقسم لأربعة أنواع حسب جودتها:

1. شرائح الشواء من لحوم حيوانات المراعى الطبيعية المفتوحة
2. شرائح الشواء من لحوم حيوانات التى تربى فى عنابر و تغذى على أطعمة طبيعية
3. شرائح الشواء من لحوم حيوانات التى تغذى على أطعمة طبيعية و أعلاف مصنعة
4. شرائح الشواء من لحوم حيوانات التى تغذى على أعلاف مصنعة

عرف كلاس تمثل (شرائح الشواء) و تمثل شحنة من نوع معين من مورد معين و تشمل مواصفات الشحنة كالسعر و الوزن و النوع و المُنتج أى المورد ، الخ و اشتق منها كلاسات تمثل الأنواع الأربعة و أضف لكل كلاس خصائص مميزة اخرى ، أضف لهذه الكلاسات بعض العمليات و setters and getters.

**لعبة سوق اللحوم الأسبانية**

فى هذه اللعبة يتم محاكاة سوق تداول اللحوم فى المدينة الأسبانية ، ففى كل يوم تأتى شحنات لحوم الشواء مجهزة من المنتجين للسوق لتباع للعملاء ، و يحدد المُنتج السعر الأدنى الذى يرغب فيه و كل مورد يحدد السعر الأدنى بشكل مستقل عن الآخرين و لكنه سعر ثابت لنفس النوع من نفس المورد و اذا لم يتقدم احد بالسعر الأدنى أو أكثر منه لا يباع اللحم ، و يعمل السوق كالتالى:

1. يعلن السوق الشحنات الموجودة و من أى موردين و كم شحنة من كل مورد و وزن كل شحنة
2. لا يعرف الترتيب الذى ستعرض به الشحنات
3. لكل شحنة يعرض اللحم و اسم المورد و السعر الأدنى
4. يقدم كل راغب فى الشراء عرضا بالسعر الذى سيدفعه و لكن هذا العرض لا يراه أو يعرفه سوى إدارة السوق
5. صاحب أعلى عرض يحصل على الشحنة
6. لكل لاعب (أى مشترى) مبلغ ثابت محدد من بداية اللعبة و كلهم متساوون فى هذا المبلغ
7. بعد انتهاء اللحوم أو النقود فإن الفائز هو من اشترى اكبر كمية من اللحم ، و إذا تساووا فالفائز هو صاحب أكبر كمية من النوع الأول ثم الثانى و هكذا.

**CS251 – Software Engineering I, 2015**

TA Name: ………………………………………… Mark: ……..

**Each student fills this form for his program and gives it to TA**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The questions to answer about each program are included in the following form. **Print and fill this form and bring with you to the discussion.**

**Student me:** ………………………………………………………………  **ID:** ………………………………….  **Group:** ……

**Which Program (Idea for task 2) did you choose?** ………………..…………………………………………………

**Which of the following Java / OOP features did you use in your program?**

1. How many classes did you create and their names? ……………………………..……………………………..
2. How many interfaces did you create and their names? ………………………………………………………..
3. How many different access specifiers did you use and their names? ……………………………………
4. How many Java coding style rules did you use and which ones? …….……………………………………

……………………………………………………………………………………………………….…………………………………….

1. How many Javadoc tags did you use and which ones? …….…………………………..………………………

……………………………………………………………………………………………………….…………………………………….

1. Did you use inheritance? When and why? …………………………………….………………..……………………

……………………………………………………………………………………………………….…………………………………….

1. Did you use method overriding? When and why? ……………………..….………………..……………………

……………………………………………………………………………………………………….…………………………………….

1. Did you use method polymorphism? When and why? ……………………..….………………..……………

……………………………………………………………………………………………………….…………………………………….

**Draw in the space below a simple UML class diagram that shows your main classes and interfaces, their attributes and operations and their interactions with each other.**

**Task 3 (2%)**

In this task, your team is required to:

1. Integrate the classes they created for Task 1 together to make an integrated working application.
2. Each team member will **explain in great details all the work** he did for the assignment including his code and documentation. He will explain the code and all related OOP and Java concepts and syntax he used. His colleagues should fully understand the code and be able to change or modify it if needed. They should be able to explain it to the TA and answer any questions about it.

**Deliver and Assessment**

1. **Submission is only accepted through acadox web site.** At least one team member should upload the work in acadox.
2. Students should store each of their programs in a separate folder.
3. Then they should provide one ZIP file named CS251\_A1Programs\_ID1\_ID…..zip
4. Each student **must** submit the form in the previous page, filled and completed. **No discussion if you do not bring your form with you.**
5. Marking will be as follows:

**Individually**

1 mark Task 1 Java program works and meets the specifications required

1 mark Use of OOP features, use of proper coding style and use of Javadoc.

1 mark Task 2 Java program works and meets the specifications required

1 mark Use of OOP features, use of proper coding style and use of Javadoc.

**Group**

2 mark If all members understand in full all the group works, they get 2/2.

**Bonus**

+ 1 mark **Outstanding** and **creative** solution.

**Policy Regarding Plagiarism:**

1. تشجع الكلية على مناقشة الأفكار و تبادل المعلومات و مناقشات الطلاب حيث يعتبر هذا جوهريا لعملية تعليمية سليمة
2. ساعد زملاءك على قدر ما تستطيع و حل لهم مشاكلهم فى الكود و لكن تبادل الحلول غير مقبول و يعتبر غشا.
3. أى حل يتشابه مع أى حل آخر بدرجة تقطع بأنهما منقولان من نفس المصدر سيعتبر أن صاحبيهما قد قاما بالغش.
4. قد توجد على النت برامج مشابهة لما نكتبه هنا أى نسخ من على النت يعتبر غشا يحاسب عليه صاحبه.
5. إذا لم تكن متأكدا أن فعلا ما يعد غشا فلتسأل المعيد أو أستاذ المادة.
6. فى حالة ثبوت الغش سيأخذ الطالب سالب درجة المسألة ، و فى حالة تكرار الغش سيرسب الطالب فى المقرر.
7. **أى مصدر تم استعماله فى بحث حالات انهيار البرمجيات يجب ذكره و لا يجب النقل المباشر و إنما الصياغة بعباراتك.**