LAU
School of Arts and Sciences

Lebanese American University
School of Arts and Sciences
Department of Computer Science and Mathematics

CSC/BIF 245 – Objects and Data Abstraction
Dr. Nadine Abbas
Fall 2021

Exam 1

Monday October 25, 2021

- The duration of this exam is **2 hours 45 minutes**.
- You may need around **10 minutes** at the end to submit your files. It is your responsibility to make sure that your files are correctly submitted without any missing or wrong files.
- The exam consists of **3 problems** for **160 points**.
- You are NOT allowed to use the web. You are NOT allowed to use USB's or files previously stored on your machine
- You are NOT allowed to communicate with a person other than the exam proctors during the exam
- You are NOT allowed to use your phones or any other unauthorized electronic devices
- The problems are of varying difficulty levels. Read through all the questions first and start with the ones that you feel the most confident about, in the order that allows you to make the most progress
- Plan your time wisely and do not spend too much time on any specific problem. If you come across a question that you can't answer leave it and move on to the next question. You can always come back to it at the end
- Submission Guidelines:
  - In this exam, you will be submitting 9 files:
    1- (2 Files) ComplexNumberTest.java, ComplexNumber.java
    2- (1 File) Problem2.java
    3- (6 Files) PersonalInfo.java, Amount.java, Account.java, BankAccount.java, LBPAccount.java, USDAccount.java
  - Make sure to include all the .java files in a folder entitled "FirstName_LastName_ID"
  - You are responsible for the correctness of your submission. It is your responsibility to make sure that your files are correctly submitted without any missing or wrong files.
- Good luck!

NAME: ____________________

ID: ____________________

Problem 1: (45 points)

In this problem, we are interested in complex numbers that can be expressed in the form $a + bi$ where a and b are real numbers representing the real *Real* and imaginary *Img* parts, respectively.

ComplexNumber
-Real: double -Img: double -Count: int

<<constructor>> ComplexNumber ()
<<constructor>> ComplexNumber (a: double, b: double)
<<constructor>> ComplexNumber (C: ComplexNumber)
//setters
//getters
+Add(C: ComplexNumber)
+isEqual(C: ComplexNumber): boolean
+Polar(double [] ArrayPolar)
+getCount(): int
+toString(): String
+finalize()

[a] **ComplexNumber class (30 points)**:
Create a class **ComplexNumber** that has the following:

- Three attributes including the real *Real* (double) and imaginary *Img* (double) parts and a static variable $Count$ (int) that increments when an object of type **ComplexNumber** is created
- A default constructor which sets the real and imaginary parts to 0
- A constructor that takes two input parameters a and b and sets the private attributes *Real* and *Img* to a and b, respectively
- Another constructor that takes as input a **ComplexNumber** C and creates the object by setting its real and imaginary values to the ones of C
- *set* and *get* methods for the *Real* and Img attributes
- *Add* method that takes as input a **ComplexNumber** C and modifies the object's real and imaginary parts to represent the sum of the complex number object and C with coordinates a and b, as follows $Real + a$ and $Img + b$, hence, adding the complex number C to the object.
- *isEqual* method that takes as input a **ComplexNumber** C and returns *true* if the object and C are equal
- *Polar* method that takes as input an array of 2 elements and sets the values of the first and second elements of the array to be the polar coordinates x and y of the complex number, respectively. The polar coordinates can be computed as follows: 1- angle $x = tan^{-1}\left(\frac{Img}{Real}\right)$ and 2- radius $y = \sqrt{Real^2 + Img^2}$
Note: You may use the Math.sqrt method that returns the square root of a number and the Math.atan method that returns the arc tangent of a number
- *getCount* a static method that returns the value of the static attribute $Count$
- *toString* method that displays the attributes in the following format "$[Real + Img\ i]$"
- *finalize* method that decrements the $Count$ and displays a message that the complex number was finalized and the value of the updated $Count$

- ***Submit your solution in a file called "ComplexNumber.java".***

[b] **Test the ComplexNumber class (15 points)**:
Write a Java program to test your class based on the sample input/output below:

```
Initially, the number of complex numbers is 0
Please enter the real and imaginary parts of A:
5
5.5
Please enter the real and imaginary parts of B:
10
6
The number of Complex Numbers is 2
After adding A and B ==> A = [15.0 + 11.5 i] and B = [10.0 + 6.0 i]
A and B are not equal!
The polar coordinates of A [15.0 + 11.5 i] are:
Angle = 0.6540827244143603
Radius = 18.9010581714358
Complex number [15.0 + 11.5 i] finalized! Count = 1
```

- ***Submit your solution in a file called "ComplexNumberTest.java".***

Problem 2: (20 points)

A **strong number** x is a number that the sum of the factorial of its digits is equal to this number. For instance, 145 is a strong number since it satisfies the following:

$$145 = 1! + 4! + 5! = 1 + 24 + 120$$

where

- 1! represents the factorial of 1,
- 4! represents the factorial of 4
- 5! represents the factorial of 5
- The factorial of a is denoted by $a!$ and is defined below.
- The sum of the factorials of 1, 4 and 5 is equal to $1! + 4! + 5! = 1 + 24 + 120 = 145$.

  - **Factorial of an integer** a, is denoted by $a!$, and defined as the product of all the positive integers less than or equal to a, i.e.,

    $$a! = 1 \times 2 \times \ldots \times (a-1) \times a$$

    If the integer a is zero, we define $a!$ to be 1, i.e., $0! = 1$. Example:

    $$\begin{aligned} \text{Factorial of } 0 = 0! &= 1 \\ \text{Factorial of } 1 = 1! &= 1 \\ \text{Factorial of } 2 = 2! &= 1 \times 2 = 2 \\ \text{Factorial of } 3 = 3! &= 1 \times 2 \times 3 = 6 \\ \text{Factorial of } 4 = 4! &= 1 \times 2 \times 3 \times 4 = 24 \end{aligned}$$

[a] ***MyFactorial* method (5 points)**
Write a method *MyFactorial* that takes as input an integer a and returns the factorial of a.
Note: You are not allowed to use any method from the Math library

[b] **3 consecutive array elements forming a 3-digit strong number (15 points)**
In this part, we are given a list of n integers less than 100, which we will be stored in an array A. We are interested in finding if there exists **3 consecutive elements** forming a 3-digit strong numbers. Accordingly, the array A should be composed of n integers, and every integer is composed of one digit, i.e., ranging between between 0 and 9.

For instance,

- 3 1 2 5 0 8, there is no sequence of 3 elements forming a 3-digit strong number.
- 1 2 $\underline{1\ 4\ 5}$ 2 1, there is a sequence of 3 elements forming a 3-digit strong number, since $1! + 4! + 5! = 1 + 24 + 120 = 145$

Your program should:

1. Prompt the user to enter the number of integers in the list n, and then prompt the user to enter n elements. In this problem, the integers in the list A should be composed of one digit (i.e., ranging between 0 and 9).
   Note: You may assume the numbers entered by the user are composed of one digit integers.
2. After filling the array A, process the array by checking whether 3 consecutive elements form 3-digit strong number. Your program should return a corresponding message as presented in the sample input/output below.
   Note: You are not allowed to use any method from the Math library. You may need to use the function $Myfactorial$ from Problem 2a

Sample input/output 1:

```
Please enter the number of integers in the list: 5
2 3 4 5 6
There is NO 3 consecutive elements forming a 3-digit strong number in the array!
```

Sample input/output 2:

```
Please enter the number of integers in the list: 6
3 1 1 4 5 6
There exist 3 consecutive elements forming 3-digit strong number in the array!
```

*** Efficient solutions are worth more grades.

- ***Submit your solution in a file called "Problem2.java".***

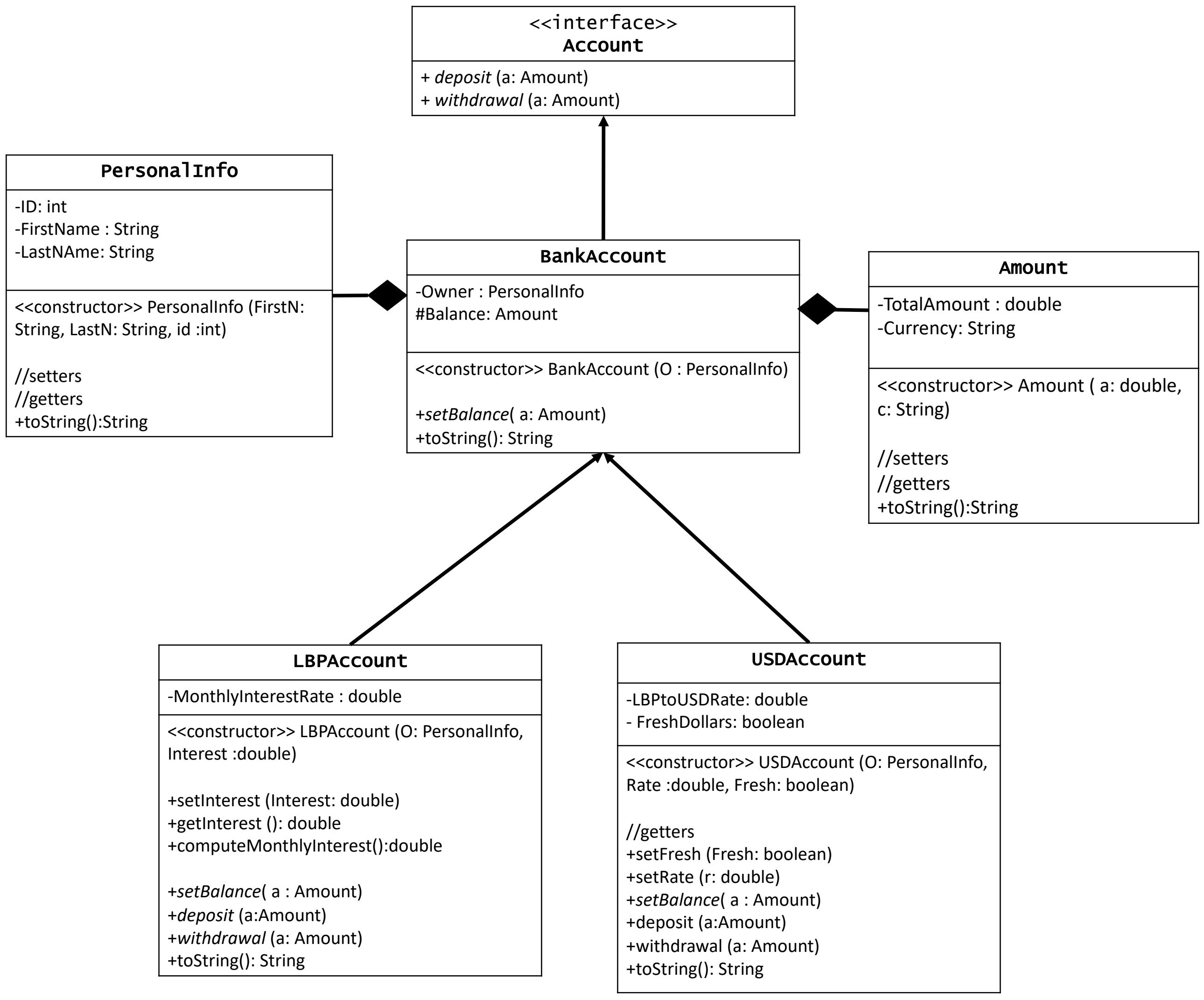
Problem 3: (95 points)

[a] **PersonalInfo class (10 points)**:
Create a class "**PersonalInfo**" as presented in the UML diagram that contains the following:

- Three attributes: ID (integer), $FirstName$ (String) and $LastName$ (String)
- A constructor with 3 arguments. You need to make sure that the ID is an integer greater than 0, otherwise set it to 0.
- *set* and *get* methods for the different attributes
- *toString* method that displays the personal information as follows:

  ```
  FirstName   LastName (ID)
  ```

- ***Submit your solution in a file called "PersonalInfo.java".***



[b] **Amount class (10 points)**:
Create a class "**Amount**" that contains the following:

- Two attributes including the total amount *TotalAmount* (double) and the currency (String) that can be either "LBP" or "USD"
- A constructor with 2 arguments
  Note: You need always to make sure that the total amount value is a positive otherwise set it to 0, and the currency is either "LBP" or "USD"; i.e.: if "LBP" set it to "LBP" otherwise set the currency to "USD".
- *set* and *get* methods for the different attributes
- *toString* method that displays the Amount as follows:

  ```
  TotalAmount in Currency
  ```

- ***Submit your solution in a file called "Amount.java".***

[c] **Account interface (5 points)**:
Implement an interface "**Account**" that contains the following two methods: *deposit* and *withdrawal* as presented in the UML diagram.

- ***Submit your solution in a file called "Account.java".***

[d] **BankAccount abstract class (10 points)**:
Create an abstract class "***BankAccount***" that implements the interface "**Account**" that contains the following:

- Two attributes including (1) a private attribute *Owner* indicating the the bank account owner information of type "**PersonalInfo**", and a protected variable *Balance* of type "**Amount**"

- A constructor with one argument the owner's info O that sets the owner info and sets the *Balance* to a new "**Amount**" object with $TotalAmount$ of 0 and $Currency$ "LBP"
- *setBalance* abstract method
- *toString* method that displays the bank account information as follows:

  ```
  Owner: FirstName   LastName (ID)
  Balance: TotalAmount in Currency
  ```

  *Note: You may need to invoke the "toString" method implemented in the "**Amount**" class*

- ***Submit your solution in a file called "BankAccount.java".***

[e] **LBPAccount class (30 points)**:
Create the first subclass "**LBPAccount**" that extends from "***BankAccount***" and contains the following:

- One private attribute representing the monthly interest rate
- A constructor that takes the owner O and the monthly interest rate $Interest$ as arguments. Your constructor should call the superclass constructor and sets the balance total amount to 0 and the currency to LBP, as well as the $MonthlyInterestRate$ to $Interest$ if the $Interest$ is greater or equal to 0, otherwise set it to 0.
- *set* and *get* methods for the interest rate attribute
- *ComputeMonthlyInterest* returns the value of the interest per month which can be computed by multiplying the monthly interest rate by the total amount of the balance
- *setBalance* method that sets the attribute *Balance* of type "**Amount**" to the new balance a
  Note: This in an LBP account, hence, you need to consider the currency when setting the new balance; i.e.: if the a amount is in "USD", the amount should be converted to LBP at a fixed conversion rate of 3900 LBP per 1 USD. You need also to keep the currency in "LBP"
- *deposit* that modifies the *Balance* total amount by adding a specific amount a of type "**Amount**" having a specific currency
  Note: As previously mentioned, this is an LBP account, the currency and the proper conversion should be taken into consideration when modifying the Balance. Ie the amount a is in "LBP", the total amount can be directly added to the Balance total amount. However, if the amount a is in "USD", you need to convert the total amount to "LBP" first and then perform the transaction.
- Similarly, *withdrawal* that modifies the *Balance* total amount by deducting a specific amount a of type "**Amount**" having a specific currency
  Note: As previously mentioned, this is an LBP account, the currency and the proper conversion should be taken into consideration when modifying the Balance. Your method should also cancel the transaction if the amount to be withdrawn is larger than the available Balance total amount.
- *toString* method that returns the information related to the LBP account
  *Note: You may need to invoke the "toString" method implemented in the "**BankAccount**"*

  ```
  * LBP Account *
  Owner: FirstName   LastName (ID)
  Balance: TotalAmount in Currency
  Monthly Interest: xx.xx
  ```

- ***Submit your solution in a file called "LBPAccount.java".***

[f] **USDAccount class (30 points)**:
Create the second subclass "**USDAccount**" that extends from "***BankAccount***" and contains the following:

- Two private attributes including the USD to LBP exchange rate *LBPtoUSDRate* and a boolean variable indicating if the USD account is an external fresh dollars account or not.
- A constructor that takes all the attributes as input except the *Balance* attribute. Your constructor should call the superclass constructor and then sets the balance total amount to 0 and the currency to *USD*. Your constructor should call the different set methods *setFresh*, *setRate* and *setBalance*, sequentially, following this specific order
- *setFresh* method that sets the attribute *FreshDollars* to the input *Fresh*. Setting *FreshDollars* to 1 indicates that the account is an external fresh dollars, and 0 otherwise.

  Note: In this problem, the LBP to USD conversion rate should be fixed and set to 3900LBP per 1 USD if the account is not a fresh account. Otherwise, the LBP to USD conversion rate is set by the LBPtoUSDRate attribute.

  Accordingly, your method *setFresh* should consider modifying the attribute *LBPtoUSDRate* based on the *Fresh* input: i.e.: If the input *Fresh* is equal to *false*, hence, the account is modified to no longer be an external fresh account, then the *LBPtoUSDRate* should be modified to be 3900LBP per 1 USD.
- *setRate* method should set the LBP to USD conversion rate to *r*. You need first to check if the rate is a positive number, otherwise set the *LBPtoUSDRate* to 3900. Moreover, the *setRate* method should consider whether the USD account is a Fresh external account or not; i.e.: If the *FreshDollars* attribute is set to *false*, i.e., the account is not a fresh external account and the *LBPtoUSDRate* should be set to 3900 LBP per 1 USD. Otherwise, the account is an external account and the *LBPtoUSDRate* can be set to the input value *r*.
- *setBalance* method that sets the attribute *Balance* of type "**Amount**" to the new balance *a*
  Note: This in a USD account, hence, you need to consider the currency when setting the new balance; i.e.: if the a amount is in "LBP", the amount should be converted to USD while taking into consideration whether the USD account is a Fresh external account and the conversion rate. You need also to set the currency to "USD".
- *deposit* that modifies the *Balance* total amount by adding a specific amount *a* of type "**Amount**" having a specific currency
  Note: As previously mentioned, this is a USD account, the currency and the proper conversion should be taken into consideration when modifying the Balance
- Similarly, *withdrawal* that modifies the *Balance* total amount by deducting a specific amount *a* of type "**Amount**" having a specific currency
  Note: As previously mentioned, this is a USD account, the currency and the proper conversion should be taken into consideration when modifying the Balance. Your method should also cancel the transaction if the amount to be withdrawn is larger than the available balance
- *toString* method that returns the information related to the USD account
  *Note: You may need to invoke the "toString" method implemented in the "**BankAccount**"*

  ```
  * USD Account *
  Owner: FirstName  LastName (ID)
  Balance: TotalAmount in Currency
  Conversion Rate: LBPtoUSDRate LBP per 1 USD
  Fresh External Account: Yes/No
  ```

- ***Submit your solution in a file called "USDAccount.java".***

NAME: ____________________

ID: ____________________