Lab: Enhanced Banking System Simulation

Objective:
In this lab, you will work in pairs to create a simple banking system simulation. The lab will focus on key OOP concepts including class creation, constructors, instance variables, methods, static variables, and string manipulation, with strict adherence to coding conventions.

With a partner, create a new project, named “comp2522 lab 1” in Jetbrains IDEA:

File / New / Project

Java / Next / Next

Project name: “comp2522 lab 1”

Finish

File / Project Structure / Modules

Create new subfolder under src / code

Create new subfolder under src / tests

OK

right-click the src/code folder / New / Java Class (then make the classes described below).

Make some <u>public</u> classes to represent a Bank and its associated classes (make their visibility "public"):

Class Name:	**Instance variables, constructor arguments, accessor methods for:**		**Methods:**

Name	first:
	last:
		String
		String
			getInitials()
			getFullName()

Date	year:
	month:
	day:
		int
		int
		int
			getYyyyMmDd()
			getDayOfWeek()
			getter methods

BankClient	name:
	date born:
	date died:
		Name
		Date
		Date
			getDetails()
			isAlive()

	client ID: signupDate:	String of 6 or 7 digits Date	getDetails()

BankAccount	client: BankClient
	balanceUsd: double
	pin: int
	accountNumber: String length 6 or 7
		deposit()
		withdraw(amountUsd)
		withdraw(amountUsd, pinToMatch)
		getDetails()

Main		main() See below

Create each of these files. Then put all of these classes into the same package, by adding this as the first line in each class:

package ca.bcit.comp2522.bank;

mouse-over the package name in each file and choose **"Move to package 'package ca.bcit.comp2522.bank;'"**

Packages are uniquely-named namespaces (essentially a "folder" of related files).

Package names must be unique, lowercase, meaningful, and in reverse domain name format).

Name class:

Create private final instance variable for first, last

Create a public constructor that takes first, last as arguments: they must not be null nor blank; must be fewer than 45 characters; must not contain the word "admin" (in any letter casing). Remember: do not use magic numbers; use constants instead.

IllegalArgumentException is thrown in case an argument is not valid

Create public accessor methods for getFirst() and getLast()

Create a getInitials() method (e.g. name of "tigER wooDS" would return "T.W.")

Create a getFullName() method (e.g. name of "tigER wooDS" would return "Tiger Woods")

Create a getReverseName() method (e.g. “tigER wooDS” would return “SDoow REgit”

Include JavaDoc comments for classes (@author and @version) and for all non-private methods and constructors (@param, @return, @throws)

The Name constructor must throw an IllegalArgumentException when its arguments are bad (null, blank, contains "admin", or more than 45 characters).

Date class

Next, let's implement the classes for the Date class:

Do not import anything into the Date class!

1) The constructor allows only years between 1800 - CURRENT_YEAR; months 1-12; and days 1-31 (or 30, or 29, or 28: properly)

2) implement the code for: getDay(), getMonth(), getYear(), getYYYYMMDD() /* returns date such as 2024-09-30 */, and getDayOfTheWeek()

To get the day of the week, do the following seven steps for dates in the 1900s:

e.g. October 31 1977:

step 1: calculate the number of twelves in 77:
    6
step 2: calculate the remainder from step 1: 77 - 12*6 = 77 - 72 =
    5
step 3: calculate the number of fours in step 2: 5/4 = 1.25, so
    1
step 4: add the day of the month to each step above: 31 + 6 + 5 + 1 =
    43
step 5: add the month code (for jfmamjjasond: 144025036**1**46): for october it is 1: 43 + 1 =
    44
step 6: add the previous five numbers: (44) and mod by 7: 44%7 = **2** (44/7 = 6 remainder 2)
step 7: sat sun **mon** tue wed thu fri is 0 1 **2** 3 4 5 6; our **2** means Oct 31 1977 was **monday**

Extra notes:
a) for January/February dates in leap years, add 6 at the start
b) for all dates in the 2000s, add 6 at the start
c) for all dates in the 1800s, add 2 at the start

Another example:

e.g. March 15 2021:

step 0: add 6 for dates in the 2000s:	NUMBER IS 6
step 1: there is 1 twelve in 21;	NUMBER IS 1
step 2: 21/12 = 1 remainder 9;	NUMBER IS 9
step 3: there are 2 fours in 9:	NUMBER IS 2
step 4:	NUMBER IS 15
step 5: month code for march is 4:	NUMBER IS 4
step 6: Add all numbers 6+1+9+2+15+4 = 37	NUMBER is 2: 37 mod 7 is 2
step 7: 37%7 = 2; 2 means monday	

Note: do not use magic numbers in your classes; instead use constants like this:

```
private static final int SATURDAY = 0;
private static final int SUNDAY = 1;
etc
```

Once you have written the code for the Date class, move onto the next classes, as follows:

1) BankClient has a Name and a birthDate and a deathDate [which can be null]; has a getDetails() method which returns a String in the exact format of "Tiger Woods (alive) was born on tuesday, December 30, 1975!" or “Albert Einstein (died monday, April 18, 1955) was born on friday, March 14, 1879!”. This class has a signupDate Date; has a String clientID which is 6 or 7 characters; update getDetails() to return a String in the exact format of "Tiger Woods client #12345 (alive) joined the bank on thursday, September 3, 2020") [note: or “not alive” as the case may be].

2) BankAccount (has overloaded methods for withdraw: withdraw(final double amountUsd); withdraw(final double amountUsd, final int pinToMatch); has a client; has an account number which is 6 or 7 characters; has a Date for accountOpened; has a Date for accountClosed which can be null if it is not closed). Provide a getDetails method that returns a String in the exact format of "Albert Einstein had $900 USD in account #abc123 which he opened on Monday January 1, 1900 and closed Saturday October 14, 1950.

5) Finally create a Main class with a public static void main method that does the following:

Create BankAccount objects for the following:

Albert Einstein (March 14, 1879 – April 18, 1955)
Print his initials, full name, reversed name, and his Person's getDetails().
He has a bank account #abc123: signed up January 1, 1900 and closed October 14, 1950.
Print his BankClient's getDetails()
His BankAccount (pin 3141) had $1000 in it, then withdraw $100.

Nelson Mandela (July 18, 1918 – December 5, 2013)
Print his initials, full name, reversed name, and his Person's getDetails().
He has a bank account #654321: signed up May 10, 1994 and is still open.
Print his BankClient's getDetails().
His BankAccount (pin 4664) had $2000 in it, then withdraw $200.

Frida Kahlo (July 6, 1907 – July 13, 1954)
Print her initials, full name, reversed name, and her Person's getDetails().
She has a bank account #frd123: signed up January 1, 1940 and closed July 13, 1954.
Print her BankClient's getDetails().
Her BankAccount (pin 1907) had $500 in it, then withdraw $50.

Jackie Chan (April 7, 1954 – still alive)
Print his initials, full name, reversed name, and his Person's getDetails().
He has a bank account #chan789: signed up October 1, 1980 and is still open.
Print his BankClient's getDetails().
His BankAccount (pin 1954) had $3000 in it, then withdraw $500.

Evaluation Criteria:

- Correctness: Proper implementation of all required features with correct use of final and units in variable names.
- Code Quality: Adherence to naming conventions, use of final for all arguments, proper commenting, and documentation using Javadocs.
- Collaboration: Effective pair programming and equal contribution from both partners.
- Comment your code well
- Do not use magic numbers anywhere in your code.
- Follow all coding standards we discussed in lecture.