Let’s review COMP2501. All of it. By creating a new project using the Jetbrains IDEA

File / New / Project

Java / Next / Next

Project name: Canada Bank

Finish

File / Project Structure / Modules

src / Sources

src / Tests

OK

right-click src folder / New / Java Class

Make some public 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 | getInitials()  getFullName() |
| Date | year  month  day | getYyyyMmDd()  getDayOfWeek() |
| Person | name date born date died | getDetails()  isAlive() |
| BankClient (extends Person) | client ID: 5 digits dateJoinedTheBank | getDetails() |
| BankAccount | client balanceUsd pin account number: 7 letters | deposit()  withdraw(amountUsd)  withdraw(amountUsd, pinToMatch) |
| Bank | ArrayList of BankAccounts | addAccount(newAccount) removeAccount(accountNumber)  getAllAccounts() // an Array  getMaxAccount() // most $  getAccountFor(client ID) |

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

**package ca.bcit.comp2601.bank;**

mouse-over the package name in each file and choose **"Move to package 'package ca.bcit.comp2601.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)

IllegalArgumentException is thrown in case an argument is not valid

Create public accessor methods

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")

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

NameTest class:

put the mouse cursor over the word "Name" in "public class Name" in Name.java

press Alt-Enter

Create Test

Class name: NameTest

Select all (four) check boxes

Fix (then wait)

Ok

OK

@Test

void getFirst()

{

Name name = new Name("tigER", "wooDS");

assertEquals("tigER", name.getFirst());

name = new Name("eLoN", "MuSk");

assertEquals("eLoN", name.getFirst());

}

right-click in that method and select **"Run 'getFirst()'"**

Now finish writing all four test methods with two assertions (minimum) as above

right-click inside the NameTest class itself (outside any of the test methods) and select **"Run 'NameTest'"** to run all of its tests.

You should see "Tests passed: 4" when you're done.

Next, let's assert that the Name constructor throws an IllegalArgumentException when its arguments are bad (null, blank, contains "admin", or more than 45 characters)

@Test

public void constructorThrows\_withBadArgument() {

Exception exception = assertThrows(IllegalArgumentException.class, () -> {

Name name = new Name(null, "wooDS");

});

String expectedMessage = "bad first name!";

String actualMessage = exception.getMessage();

assertTrue(actualMessage.equals(expectedMessage));

}

Now add in more tests for empty first name, first name of whitespace only, first name containing admin, first name more than 45 characters long...then repeat for last name.

# Date class

Next, let’s implement the classes and tests for the Date class:

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

2) implement the code and tests for: getDay(), getMonth(), getYear(), getYYYYMMDD() /\* returns date such as 2022-09-30 \*/, 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 up: 44; mod that number 44 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 number 2 means October 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

step 7: 37%7 = 2; 2 means monday

Note: do not use magic numbers in your classes (it's ok to use them in tests); 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 and DateTest class, move onto the next classes, as follows:

1) Person (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!”.

2) BankClient (extends Person; has a dateJoinedBank Date; has a String clientID which is 5 digits; overrides 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].

3) 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 7 letters; has a Date for accountOpened; has a Date for accountClosed which can be null if it is not closed)

4) Bank (has a CEO who is a Person; has an ArrayList of BankAccount references; addAccount() throws an IllegalArgumentException if an account is being added but its accountNumber is already in the Bank's account collection; getCeo() returns the current CEO)