Work on this assignment by yourself.

Write the code for the assignment below and upload the zipped project folder to the learning hub (learn.bcit.ca → Activities → Assignments →Assignment 2) before the deadline (a few days before lesson 12).

Assignment 2 tests everything we have learned from lessons 7 to 11: unit testing, GUI, streams/filters, design patterns, and concurrency.

Include your full name at the top of each file, using a Javadoc comment: for example:
/ @author Tiger Woods */**

Do not use magic numbers in your code. Use all of the best practices we have used in class.

Create a Main class with the main() method which shows a menu of options for the user to choose from. It shows the following screen and calls the chosen method as listed; then the menu appears again…repeat until the user types 6 to quit:

Type 1 for GUI	// displays CountryList's window
Type 2 for Streams and Filters	// calls CountryProcessor class's processCountries() method
Type 3 for Design Patterns	// calls Tester class's test() method
Type 4 for Concurrency	// calls concurrentMain() from your class (see below)
Type 5 to Quit	// ends the program (do not call **System.exit()**; just end the menu loop)

Lesson 7: Unit Testing

Create a proper JUnit test named **PersonTest** that replaces (but does the same job as) the testPerson() test from the **Assignment1Tester.java** file. Recall that the testPerson() method provided by your instructor for assignment 1 was not a proper JUunit test…it was just an informal type of test. Test everything that assignment1's testPerson method did. Include your Person class (or even the sample solution's Person class) with assignment 2 so that when your instructor runs your new PersonTest unit test, the Person class passes all your tests.

Lesson 8: GUI

In a class called **CountryList**, displays all countries and their capitals in a JList (e.g. Canada: Ottawa), in alphabetical order by country name. Use the data from the **countries.txt** file. When the user closes the window, show the Main.main() menu.

Lesson 9: Streams and Filters

In a class called **CountryProcessor**, create a function named **processCountries()** that calls all eight of the following functions which you will write. Use the data from the **countries-and-capitals.txt** file. Create a HashMap instance variable with country name as key, and its capital city name as value (e.g. **"Canada": "Ottawa"**). Use streams and filters to create the following functions which do exactly what they say. In all cases, before printing, collect the result into a java collection local variable:

1. printLongestCapitalCity()
2. printShortestCountryName()
3. printAllCountriesStartingWith(String substring)

4. printLongestCombination() // longest combination of country name plus capital city name
5. printHowManyLettersInCountries() // the total number of letters in all the country names put together
6. printCapitalsWithThisManyLetters(int min, int max) // e.g. all capitals between 5 and 8 letters inclusive
7. printAllCountriesThatDoNotEndWith(char letter)
8. printAllCapitalsThatContainLetterIntoASingleStringNoSpaces(char letter) // e.g. containing 'a': "CanadaChadArgentinaNewZealandAustralia…"

Lesson 10: Design Patterns

Create a class named **Tester**, with a method named **test()**. Read the notes provided by your instructor (Singleton), your team, and the other teams (Adapter, Command, Observer) and make the following five very-small classes:

a) Singleton: Make a **PrimeMinister** class. There can only be one Prime Minister object at a time, so it must implement the Singleton design pattern. The **Tester.test()** method must try to create four PrimeMinister objects, yet the Singleton will create only one; the others will simply be references to the first. **Tester.test()** must print all four objects to show they all actually reside at the same memory address.
b) Adapter: Literally implement the code at https://www.baeldung.com/java-adapter-pattern. The **Tester.test()** method will show that your **BugattiVeyron** converts MPH to KMPH by calling its **getSpeed()** method and showing it's within 0.00001 of 431.30312 KMPH when its speed is set to 268 MPH.
c) Command: Literally implement the code at https://www.baeldung.com/java-command-pattern. You may choose OOP or functional (i.e. sections 2 or 3 in the code there). Your **Tester.test()** will run the code listed on this website's main() method.
d) Observer: Literally implement the code in steps 1 and 2 at https://www.baeldung.com/java-observer-pattern. **Tester.test()** will run the code at the end of step 2.

Lesson 11: Concurrency

Implement any one of the code samples at https://www.zghurskyi.com/concurrent-sum-of-numbers/. Instead of putting their main() method code into main(), rename the method "**concurrentMain()**" and call it from the menu when the user chooses **Type 4 for Concurrency**.

Zip your project folders and upload the zip file before the due date. The zip file must include the following files:

countries.txt	(unmodified)
countries-and-capitals.txt	(unmodified)
Main.java	(with main() method)
Person.java	(yours or even your instructor's, from assignment 1)
PersonTest.java	(with testPerson() unit test)
CountryList.java	
CountryProcessor.java	(with processCountries() and 8 other functions)
Tester.java	(with test() method) which also uses these four classes:
- PrimeMinister.java	
- BugattiVeyron and related interfaces/classes	
- TextFileOperation interface and related interfaces/classes	
- NewsAgency class and related classes	
DivideAndConquerSum.java	(with concurrentMain() method, plus all the methods you chose from the site)