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 1) before lesson 9 begins.

Assignment 1 tests everything we have learned from lessons 1 to 7: abstract classes, inheritance, polymorphism, interfaces, class initialization, lambda expressions, method references, exception handling, input/output streams, and unit testing.

The **Assignment1Tester.java** file that has been provided will help you determine whether your code is meeting the requirements. Use it to help guide you as you design and code the program.

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.

The provided Assignment1Tester.java will use **2022** as the current year; you should do the same (details below).

Create the following types and put them into a well-named package of your own choosing.

Interfaces:

1. Orderable
2. Writeable (a Functional Interface)

Classes:

3. Name
4. Date
5. IllegalPersonException (extends RuntimeException)
6. Person
7. Student (extends Person)
8. Teacher (extends Person)
9. School

The following pages describe the requirements for each interface and class.

Orderable Interface

This interface has two abstract methods:

public Orderable next()

public Orderable previous()

Writeable Interface

This functional interface has one abstract method:

public void printDate(String s, int min, int max)

Name class

This class has final instance variables, constructor arguments, and accessor methods for **String first** and **String last**. It has the following methods:

public Name(String first, String last): This constructor throws IllegalArgumentException exceptions if either **first** or **last** is null or blank.

public String getPrettyName(): this method returns a formatted version of the full name; e.g. if **first** is "tiGEr" and **last** is "woODs", this method returns "Tiger Woods".

public String getInitials(): this method returns the initials of the full name; e.g. if **first** is "tiGEr" and **last** is "woODs", this method returns "T.W.".

Date class

This class implements the **Orderable** and the **Comparable** interfaces (details below).

This class has final instance variables, constructor arguments, and accessor methods for **int day**, **int month**, and **int year**. It has the following methods:

public Date(int day, int month, int year): This constructor throws IllegalArgumentException exceptions if the year is 0, if the month is not 1 – 12 (remember, though, to avoid using magic numbers), or if the day is not appropriate for the month (e.g. February has only 28 days, February has 29 days in leap years, March has 31 days, April has 30 days, etc…).

public String getYyyyMmDd(): this method returns a String representation of the Date in "yyyy-mm-dd" format (e.g. Christmas Day 2032 returns "2032-12-25").

This class overrides the **public String toString()** method, which returns the value from its own **public String getYyyyMmDd()** method.

public Date previous() and **public Date next()**: these two methods satisfy the requirements from implementing the **Orderable** interface. If the current date were January 1 2000, then **previous()** must return December 31 1999; **next()** would return January 2 2000, etc….

public int compareTo(Date d): this method satisfies the requirements from implementing the **Comparable** interface. Later dates are "larger".

public String getDayOfTheWeek(): this method determines and returns the day of the week for a given date. It must use the algorithm given here. It could be useful to create some private methods (e.g. **private int getNumberOfDaysPerMonth(int month, int year)**, etc…). Here is the algorithm you must implement:

This method returns the day of the week (e.g. "Wednesday") for a specified date (e.g. October 31, 2012).

It must make use of a private method **private boolean isLeapYear()**
(See: http://en.wikipedia.org/wiki/Leap_year)

This method returns **true** (e.g. for 2000, 2024, 2028, etc…) or false (e.g. for 1900, 2023, etc…) depending on whether a year is a leap year.

Here is the algorithm to determine what day of the week a given date is:

Two Example dates: August 16, 1989 and March 20, 1950

Step 1: Only look at the last two digits of the year and determine how many twelves fit in it:

7 twelves in 89 4 twelves in 50

Step 2: Determine the remainder of step 1's result:

89 – (7 * 12) = 5 50 – 4 * 12 = 2

Step 3: Determine how many fours fit into the remainder (step 2's result):

There is 1 four in 5 There are 0 fours in 2

Step 4: Add the day of the month:

16 for August 16th 20 for March 20th

Step 5: Add the month code from the table below (hint: writing a private method to determine this would be a good idea: **private int getCodeForMonth(int month)**):

3 for August 4 for March

Jan = 1	Feb = 4	Mar = 4
Apr = 0	May = 2	Jun = 5
Jul = 0	Aug = 3	Sep = 6
Oct = 1	Nov = 4	Dec = 6

Step 6: Add all of the above highlighted numbers, and then mod by 7:

7 + 5 + 1 + 16 + 3 = 32 4 + 2 + 0 + 20 + 4 = 30
32 % 7 = 4 30 % 7 = 2

That is your day of the week, as follows:

Sat = 0	Sun = 1	Mon = 2	Tue = 3	Wed = 4	Thu = 5	Fri = 6

August 16, 1989 March 20, 1950
Wednesday Monday

NOTE: some dates require special offsets to add after step 5:

January and February dates in leap years: add 6 to step 5

All dates in the 1600s: add 6 to step 5
All dates in the 1700s: add 4 to step 5
All dates in the 1800s: add 2 to step 5
All dates in the 2000s: add 6 to step 5
All dates in the 2100s: add 4 to step 5

IllegalPersonException class

This class extends the **RuntimeException** class. It satisfies its parent's constructor and does nothing else.

Person class

This class implements the **Comparable** interface (details below).

This class has instance variables and accessor methods for **Date born**, **Date died**, and **Name name**. It has the following methods:

public Person (Date born, Name name): This constructor throws **IllegalPersonException** exceptions if either **born** or **name** is null.

public void die(Date dateOfDeath): this method sets the **died** instance variable to the **dateOfDeath** provided.

public boolean isAlive(): this method returns true if the Person is alive; otherwise returns false.

public int compareTo(Person p): this method satisfies the requirements from implementing the **Comparable** interface. Younger people are "larger". Note: this method must use its **born** variable's **compareTo(Date d)** method.

This class overrides the **public String toString()** method, which returns a String in one of these two exact formats:

a) Alive people example: "Tiger Woods was born 1975-12-30 and is still alive"
b) Dead people example: "Albert Einstein was born 1879-03-14 and died 1955-04-18"

Use the **name** variable's **getPrettyName()** method, and the **born**/**died getYyyyMmDd()** method.

Student class

This class extends the **Person** class. It also has an additional instance variable and its accessor method: **String studentNumber** and **public String getStudentNumber()**.

public Student (Date born, Name name, String studentNumber): This constructor throws **IllegalPersonException** exceptions if **born** or **name** is null; also, it throws **IllegalPersonException** exceptions if **studentNumber** is null, blank, or not nine characters long.

This class overrides the **public String toString()** method, which returns a String in one of these two exact formats:

a) Alive people example: "Tiger Woods (student number: A12345678) was born 1975-12-30 and is still alive"
b) Dead people example: "Albert Einstein (student number: A87654321) was born 1879-03-14 and died 1955-04-18"

Use the parent's **getName()** return value, then its **getPrettyName()** method, and the **born**/**died getYyyyMmDd()** method.

Teacher class

This class extends the **Person** class. It also has an additional instance variable and its accessor method: **String specialty** and **public String getSpecialty()**.

public Teacher (Date born, Name name, String specialty): this constructor throws **IllegalPersonException** exceptions if **born** or **name** is null. Also, it throws an **IllegalPersonException** exception if **specialty** is blank (note: null is ok).

This class overrides the **public String toString()** method, which returns a String in one of these two exact formats:

a) Alive people example: "Tiger Woods (specialty: mathematics) was born 1975-12-30 and is still alive"
b) Dead people example: "Albert Einstein (specialty: mathematics) was born 1879-03-14 and died 1955-04-18"

Use the parent's **getName()** return value, then its **getPrettyName()** method, and the **born/died getYyyyMmDd()** method.

School class

There is one instance variable for this class: a **List of Person references**.

Use an initializer block to create an empty ArrayList and assign it to your List variable.

public void register(Person p): this method throws an **IllegalPersonException** if the argument is null. Otherwise, the **Person** is added to the end of your List.

public void printRoster(): this method uses a method reference to print out all the people (i.e. triggering a call to their respective **toString()** methods).

public void printAgesAndYears(): this method declares a local variable **w** of type **Writeable**, which takes three parameters: **fullName**, **yearBorn**, and **maxYear**; for example: "Tiger Woods", 1975, 2022, or "Albert Einstein", 1879, 1955. It uses a lambda expression to loop through the integers from **yearBorn** to **maxYear** and prints the person's **name** and **age** for each **year of life**. For example (use all the years; some were omitted here for brevity):

```
Tiger Woods: 1975 (age 0)
Tiger Woods: 1976 (age 1)
Tiger Woods: 1977 (age 2)
Tiger Woods: 1978 (age 3)
Tiger Woods: 1979 (age 4)
Tiger Woods: 1980 (age 5)
… etc …
Tiger Woods: 2019 (age 44)
Tiger Woods: 2020 (age 45)
Tiger Woods: 2021 (age 46)
Tiger Woods: 2022 (age 47)          // if the Person is alive, loop until the current year
```

Finally, loop through your List of Person references and use a lambda expression to determine the argument values to pass to **w's printData()** method - **fullName**, **yearBorn**, and **maxYear** – as follows:

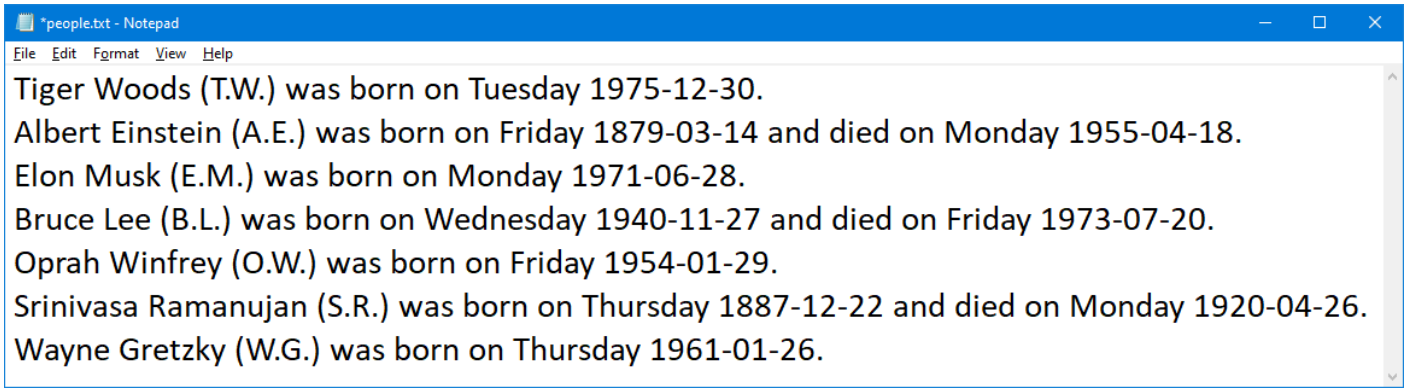
1. fullName: the Person's getName().getPrettyName()
2. yearBorn: the Person's getDateOfBirth().getYear()
3. maxYear: the current year (if alive), or the Person's getDateOfDeath().getYear() (if not alive).

```
Albert Einstein: 1879 (age 0)
Albert Einstein: 1880 (age 1)
Albert Einstein: 1881 (age 2)
Albert Einstein: 1882 (age 3)
… etc …
Albert Einstein: 1953 (age 74)
Albert Einstein: 1954 (age 75)
```

Albert Einstein: 1955 (age 76) // if the Person is not alive, loop until their last year of life

Note: there is an example of how to create and use a **Writeable** lambda expression, in the **testWriteable()** method of the **Assignment1Tester.java** file.

public void saveDetails(): this method writes data from all the School's people, to a file called "people.txt" in the following format:



The Assignment1Tester.java file will give you a score out of 110, which is a rough guideline of how complete your code is. More marks are given for style and following best practices and conventions.