This is the first part of Deliverable 1. It is a review of your previous OOP courses. This needs to be completed as it is the foundation for all of the deliverables going forward. You are encouraged to ask questions during lab periods if you are unclear on any of the requirements.

All classes should exist in a *webd4201.lastnamefirstinitial* package (all lowercase and *lastnamefirstinitial* should be your last name and first initial).

Note: the classes and all attributes/methods/constructors in your project files must have Javadoc comments for full marks
1 mark penalty per class, interface, constructor, method, attribute that is missing Javadoc comments
0.5 mark penalty per class, interface… that Javadoc comments are incomplete

1. Create a Java interface named *CollegeInterface.java*, this interface should:
    a. include constants named:
        i. `COLLEGE_NAME` of type String that stores "`Durham College`"
        ii. `PHONE_NUMBER` of type String that stores "`(905)721-2000`"
        iii. `MINIMUM_ID_NUMBER` of type long (primitive) that stores `100000000`
        iv. `MAXIMUM_ID_NUMBER` of type long (primitive) that stores `999999999`
    b. a method header for a method called *getTypeForDisplay()* that takes no arguments but returns a String. This method should be set up so that it is accessible anywhere and an instance method (not a class method).

2. Create a `User.java` class will be used to create objects of the `User` type, this class should have the following:

    a. Be coded so that you cannot instantiate objects of this type with the *new* keyword

    b. Be coded so that it implements the above *CollegeInterface.java*

    c. Has shared class constants (NOTE: all of these constants should be set so they accessible to any subclasses, whether they are in the same package or not)
        i. `DEFAULT_ID` of type long that stores `100123456`
        ii. `DEFAULT_PASSWORD` of type String that stores "`password`"
        iii. `DEFAULT_FIRST_NAME` of type String that stores "`John`"
        iv. `DEFAULT_LAST_NAME` of type String that stores "`Doe`"
        v. `DEFAULT_LAST_NAME` of type String that stores "`john.doe@dcmail.com`"
        vi. `DEFAULT_ENABLED_STATUS` of type boolean that stores "`true`"
        vii. `DEFAULT_TYPE` of type char that stores '`s`'
        viii. `ID_NUMBER_LENGTH` of type byte that stores `9`
        ix. `DF` of type DateFormat that formats dates to the `DateFormat.`*`MEDIUM`* for `Locale.`*`CANADA`*

d. Have instance attributes named (NOTE: all of these attributes should be set so they are only accessible from inside the class):
    i. `id` of type long to store the user's unique identification number
    ii. `password` of type String to store the user's password
    iii. `firstName` of type String to store the user's first name
    iv. `lastName` of type String to store the user's last
    v. `emailAddress` of type String that will be used to store the user's email address.
    vi. `lastAccess` of type Date to store the last time the user accessed the system
    vii. `enrolDate` of type Date to store when the user was created in the system
    viii. `enabled` of type boolean to store whether the user is enabled in the system (false would mean the user is disabled)
    ix. `type` of type char to store the user's type as a single character

e. For each of the instance attributes created above, you are to create correctly name getter(accessor) and setter(mutator) methods. These accessor/mutator methods should set up so that they are accessible from anywhere. NOTE: you can do this simply in Eclipse by creating the attributes and then selecting *Source -> Generate Setters and Getters…* and accepting the defaults.

f. constructors (that should be accessible from anywhere)
    i. a parameterized one that accepts arguments, one each for the types corresponding to the attributes listed above, that places each argument into it's appropriate attribute using the `setXxx()` method

    ii. a default one (i.e. one that does not take any arguments), that will set the instance attributes to the public class attributes created above. NOTE: set the enrol date and last access date to today (i.e. the day the program was run). For full marks this default constructor should called the parameterized constructor created above using the `this` keyword (passing the class attributes and today's date, using `new Date()` for those two arguments, twice).

g. Create a instance method named `toString()` that overloads the `java.Object`'s `toString()` creates a specific user's information as a String in the form:

```
User Info for: 100123456
     Name: John Doe
     Created on: 15-Jan-2016
     Last access: 15-Jan-2016
```

h. Create an instance method named `dump()` that does not take any arguments, returns nothing and just displays the returned string from the `toString()` method using `System.out.println(toString()).` This method should be available from anywhere.

i. Have a class (i.e. `static`) method named `verifyId()`that accepts a `long` argument and checks, using the minimum and maximum values for a valid id, and returns a boolean

3. You will create a `Faculty.java` class file that will contain the following:
   a. class attributes named (NOTE: their access type should be `public`, and they should be set up so that they cannot be changed/are constants)
      i. `DEFAULT_SCHOOL_CODE` of type `String` that stores "`SET`"
      ii. `DEFAULT_SCHOOL_DESCRIPTION` of type `String` that stores "`School of Engineering & Technology`"
      iii. `DEFAULT_OFFICE` of type `String` that stores "`H-140`"
      iv. `DEFAULT_PHONE_EXTENSION` of type `int` that stores `1234`

   b. Inherits all of the methods and attributes of the `User` class, but that also has the following instance attributes (declared in such a way that they can only be accessed from inside the class):
      i. a `String` named `schoolCode` that will store a code for the school the faculty is associated with. E.g. BITM for School of Business, IT & Management
      ii. a `String` named `schoolDescription` that will store a code for the school the faculty is associated with. E.g. "School of Business, IT & Management"
      iii. a `String` named `office` that will store the faculty member's office location. E.g. BITM for School of Business, IT & Management
      iv. a primitive `int` named `extension` that will store the faculty member's phone extension

   c. For each of your new attributes create accessor/mutator methods. Make these public accessible.

   d. Defines the following constructors:
      i. a parameterized constructor that takes one argument each for the inherited and new attributes, then calls the User parent's parameterized constructor using the `super()` keyword, passing the user arguments provided, and then sets the extra `Faculty` specific attributes using the appropriate `setXxx()`methods

      ii. a default constructor should call the parent's default constructor (using the `super()` call) and set the extra attributes using the appropriate `setXxx()`methods

e. Implements the abstract `getTypeForDisplay()` method so that is simply returns the word "Faculty"

f. Override the base class' `toString()` method so that the Faculty's info is displayed as below. NOTE; for this method the first part of the output is almost identical to the `User`'s `toString()` output, you are therefore required to call the parent's method explicitly (placing the returned String into a local variable), replace the word "User" with the word "Faculty", and then add the faculty specific info to the end of the output using the `getXxx()` methods:

```
Faculty Info for: 100123456
     Name: John Doe (john.doe@dcmail.com)
     Created on: 28-Jan-2016
     Last access: 28-Jan-2016
     School of Business, IT & Management (BITM)
     Office: H-140
     (905)721-2000 x1234
```

NOTES: the word "Faculty" should come from the `getTypeForDisplay()` method; to replace part of a string with something different, Java provides a `replaceAll()` method as part of the `String` class. To replace the word "User" with the word "Faculty" the call would be:

```
output = output.replaceAll("User", "Faculty");
```

4. Create a `Mark.java` class file that:
   a. *class* attributes named (NOTE: their access type should be `public`, and they should be set up so that they cannot be changed/are constants)
      i. `MINIMUM_GPA` of type float that stores 0.0
      ii. `MAXIMUM_GPA` of type float that stores 5.0
      iii. `GPA` of type DecimalFormat that formats decimal numbers to one decimal place with a leading zero
   b. has the following attributes (declared in such a way as that they are only accessible from inside the c lass):
      i. a string named `courseCode` attribute that will contain the course code for the mark
      ii. a string `courseName` that will contain the course name for the mark
      iii. a primitive integer named `result` that will store the student's final result in the course
      iv. a float named `gpaWeighting` that will hold the course's GPA weighting
   c. has accessor/mutator methods for this class (that are accessible from everywhere)

d. has a parameterized constructor that takes arguments, one each for the attributes above (this constructor should be accessible from everywhere). The constructor should use the `setXxx()` methods.
e. Has a `toString()` method that returns a String of the mark's info. Be sure to use the `getXxx()` methods to retrieve the objects attributes. The output should take the form:

```
WEBD2201   Web Development - Fundamentals     71       4.0
```

NOTE: the GPA weighting should be formatted to show one decimal place and a leading zero using the class `GPA` DecimalFormat object.
To set a consistent width for the course name, use the `String.format()` method, this will allow to set a fixed width. The following will output the course name 35 characters wide (the negative sign left-aligns the output):

```
String.format("%-35s" , getCourseName())
```

5. Create a `Student.java` class file that:
   a. class attributes named (NOTE: their access type should be `public`, and they should be set up so that they cannot be change/are constants)
      i. `DEFAULT_PROGRAM_CODE` of type String that stores "`UNDC`"
      ii. `DEFAULT_ PROGRAM _DESCRIPTION` of type String that stores "`Undeclared`"
      iii. `DEFAULT_YEAR` of type int that stores 1 (default students will be first years)

   b. Inherits all of the methods and attributes of the `User` class, but that also has the following instance attributes (declared in such a way as that they are only accessible from inside the class):
      i. A String named `programCode` , that will store the program code for the Student object: CSTC for Computer Systems Technician; CSTY for Computer Systems Technology; CPGM for Computer Programmer; CPA for Computer Programmer Analyst etc.
      ii. A String named `programDescription`, that will store the program description for the Student object: "Computer Systems Technician"; "Computer Systems Technology"; "Computer Programmer"; " Computer Programmer Analyst" etc.
      iii. A int primitive named `year` that will hold the year of the program they are currently enrolled in 1 for first year, 2 for second year, 3 for third year.
      iv. A Vector of Mark objects named `marks`

   c. For each of your new attributes create accessor/mutator methods. Make these accessible from anywhere.

   d. Defines the following constructors:

i. a parameterized constructor that takes one argument each for the inherited and new `Student` attributes. This constructor should call then sets each attribute using the different `setXxx()` methods (the public ones in the parent, and new methods in the derived class) NOTE: you are to call the base class' constructor using the `super()` keyword explicitly passing the User at.

ii. An overloaded constructor that takes arguments for all attributes for a Student object EXCEPT for a Vector of `Mark` objects. This constructor should call the above constructor using the this keyboard, sending all of its arguments PLUS an empty Vector of Mark objects.

iii. a default constructor should calls the parameterized one defined that does not include a Vector of `Mark` objects passing all of the default values using the `this()` keyword (including the one's found in the parent), you are NOT to call the super() constructor explicitly.

```
this(DEFAULT_ID, DEFAULT_PASSWORD, DEFAULT_FIRST_NAME,
        DEFAULT_LAST_NAME, DEFAULT_EMAIL_ADDRESS,
        new Date(), new Date(), DEFAULT_TYPE,
        DEFAULT_ENABLED_STATUS, DEFAULT_PROGRAM_CODE,
        DEFAULT_PROGRAM_DESCRIPTION, DEFAULT_YEAR);
```

e. Override the base class' `toString()` method so that the Student's info is displayed as below. For this method have all attributes retrieved using the `getXxx()` methods. You are not to call the parent's `toString()` method (as it is totally new output):

Example 1:

```
Student Info for:
     John Doe (100123456)
     Currently in 1st year of "Undeclared" (UNDC)
     Enrolled: 23-Jan-2017
```

Example 2:

```
Student Info for:
     YOUR_FIRST_NAME YOUR_LAST_NAME (100543210)
     Currently in 2nd year of "Computer Programmer Analyst" (CPA)
     Enrolled: 4-Sep-2017
```

NOTES: the suffix changes for 1st, 2nd and 3rd years (a student in their 4th year should have a "th" suffix)

6. Create the following specialized exception classes (by `extend`'ing the generic `Exception` class). Your exceptions should overload both the default constructor (no arguments) and the overloaded one that takes a single String argument for a message)
    a. `InvalidIdException` to be used with the `setId()` method (and subsequently the constructors that use the `setId()` method)
    b. `InvalidPasswordException` to be used with the `setPassword()` method (and subsequently the constructors that use the `setPassword()` method)
    c. `InvalidNameException` – this exception will be with the `setFirstName()` and `setLastName()` methods (and subsequently the constructors that use the `setFirstName()` and `setLastName()` methods)
    d. `InvalidUserDataException` – this exception will be used to indicate if any passed data to create a User object is incorrect (i.e. will be used to re-package the above exceptions in the User constructors)

    NOTE: you are to include code that will remove the IDE provided warning:

    ```
    The serializable class InvalidIdException does not declare a static
    final serialVersionUID field of type long
    ```

    This can be achieved:

    ```
    @SuppressWarnings("serial")
    ```

    at the top of EACH exception class or declare in each exception file a class variable named serialVersionUID:

    ```
    private static final long serialVersionUID = 1L;
    ```

    e. Set up logical exception handling, this should include detailed messages in the exceptions that are thrown in your `User`, `Faculty` and `Student` classes. Things that should be handled:
        i. the `id` being set is not between the minimum and maximum values
        ii. the `password` being set is too long or too short
        iii. the `firstName` being set cannot be an empty String or a number
        iv. the `lastName` being set cannot be an empty String or a number