Concordia University
Comp 249 – Summer 2020
Object-Oriented Programming II
Assignment 1
Due 11:59 PM – Sunday , July 12 , 2020

Purpose: The purpose of this assignment is to practice class inheritance, and other Object-Oriented Programming concepts such as constructors, access rights, method overriding, etc. The assignment would also allow you to practice the notion of package creation. This assignment contains two parts. You need to complete Part I to be able to do Part II.

Part I

Various public transportation can be described as follows:

- A **PublicTransportation** class with the following: *ticket price* (double type) and *number of stops* (int type).
- A **CityBus** is a **PublicTransportation** that in addition has the following: an *route number* (long type), an *begin operation year* (int type), a *line name* (String type), and *driver(s)name* (String type).
- A **Tram** is a **CityBus** that in addition has the following: *maximum speed* (int type), which indicates the maximum speed that this Citybus could reach.
- A **Metro** is a **CityBus** that in addition it has the following: *number of vehicules* (int type) and *the name of the city* (String type), such as Montreal, Toronto, etc.
- A **Ferry** is a **PublicTransportation** that in addition has the following: *build year* (int type) and *Ship name* (String type).
- An **AirCraft** is a **PublicTransportation** that in addition has the following: *class type* (enumeration type that can be: *Helicopter, Airline, Glider* or Balloon), and *maintenance type* (enumeration type that can be: *Weekly*, *Monthly*, or *Yearly*).

Part I:

1. Draw a UML representation for the above-mentioned classes hierarchy. Your representation must also be accurate in terms of UML representation of the different entities and the relation between them. You must use software to draw your UML diagrams (no hand-writing/drawing is allowed).

2. Write the implementation of the above-mentioned classes using inheritance and according to the specifications and requirements are given below:

   - You must have 4 different Java packages for the classes. The first package will include the **PublicTransportation** class. The second package will include the **CityBus**, **Tram,** and **Metro** classes. The third package will include the **Ferry** class and the last package will include the **AirCraft** class.

- For each of the classes, you must have at least three constructors, a default constructor, a parametrized constructor (which will accept enough parameters to initialize ALL the attributes of the created object from this class), and a copy constructor. For instance, the parametrized constructor of the **Tram** class must accept 7 parameters to initialize the *price*, the *number of stops*, the *route number*, the *begin operation year*, the *line name*, the *driver(s)name*, and *maximum speed.*

- An object creation using the default constructor must trigger the default constructor of its ancestor classes, while creation using parametrized constructors must trigger the parameterized constructors of the ancestors.

- For each of the classes, you must include at least the following methods: accessors, mutators, ***toString()*** and ***equals()*** methods (notice that you are always overriding the last two methods).

  - The ***toString()*** method must display a clear description and information of the object (i.e *"This Tram has 23 stops, and costs 17$. Its maximum speed is 70km/h….."*).

  - The ***equals()*** method must first check if the passed object (to compare to) is null and if it is of a different type than the calling object. The method would clearly return false if any of these conditions is true; otherwise, the comparison of the attributes are conducted to see if the two objects are actually equal. You need to add a comment indicating how effective these null verifications inside the method will be in relation to protecting your program from crashing!

- For all classes, with the exception of the **CityBus** class, you **must** to use the appropriate access rights, which allows most ease of use/access without compromising security (Do not use most restrictive rights unless they make sense!). For the **CityBus** class for example, you are required to use protected access rights.

- When accessing attributes from a base class, you must take full advantage of the permitted rights. For instance, if you can directly access an attribute by name from a base class, then you must do so instead of using a public method from that base class to access the attribute.

3. Write a driver program (that contains the main() method) that would utilize all of your classes. The driver class can be in a separate package or in any of the already existing four packages. In the main() method you must:
   - Create various objects from the 6 classes, and display all their information using the ***toString()*** method.
   - Test the equality of some to the created objects using the ***equals()*** method.
   - Create an array of 15 **PublicTransportation** objects and fill that array with various objects from the 6 classes (each class must have at least one entry in that array).
   - Trace(search) that array to find the object that is least expensive (has the lowest price) and the one that is most expensive. Display all information about these two objects along with their location (index) in the array.

Part II

For the requirements of this part, you need to modify your implementation from Part I as follows:

1. The classes from Part I must now have the most restrictive (secure/protective) access rights to their attributes. Adjust your implementation from Part I accordingly, and comment on the decision about using restricted access (i.e. what are the tradeoffs).

2. In the driver program of this part, you need to add to the one from Part I, another static method (added it above the main()method), called ***copyCityBuss***. The method will take as input an array of **PublicTransportation** (an array of any size) and returns an array of **PublicTransportation**. That is to say, the method needs to create an array of the same length as the passed parameter one, copy all CityBuss from the passed array to a new array then return it. Your copy of the objects **must** use the copy constructors of the different listed classes. (Please read point # 4 below carefully!)

3. In the driver program, create an array of 12 objects (must have at least one from each of the classes), then call the *copyCityBuss()* method to create a copy of that array.

4. Display the contents of both arrays, then add some comments indicating **whether or not the coping is correct. If not; you need to explain why it has not been successful or as you might expect. Additionally, if the copying is not correct, do NOT try to correct it. You rather need to comment explicitly on the reasons for such misbehavior.**

JavaDoc Documentation:

Documentation for your program must be written in **JavaDoc**.
In addition, the following information must appear at the top of each file:

```
Name(s) and ID(s)   (include full names and IDs)
COMP249
Assignment #        (include the assignment number)
Due Date            (include the due date for this assignment)
```

General Guidelines When Writing Programs

- Include the following comments at the top of each class you are writing.

```
// -----------------------------------------------------
// Assignment (include number)
// Part: (include Part Number)
// Written by: (include your name(s) and student ID(s))
// -----------------------------------------------------
```

- Use appropriate comments when needed.
- Display clear prompts for the user whenever you are expecting the user to enter data from the keyboard.
- All outputs should be displayed with clear messages and in an easy to read format.
- End your program with a closing message so that the user knows that the program has terminated.

Submitting Assignment 1

- For this assignment, you are allowed to work individually, or in a group of a maximum of 2 students (i.e. you and one other student). You and your teammate must however be in the same section of the course. Groups of more than 2 students = zero mark for all members!
- Only electronic submissions will be accepted. Zip together with the source codes.

- Students will have to submit their assignments (one copy per group) using the Moodle system (please check for your section submission).
- Naming convention for zip file: Create one zip file, containing all source files and produced documentation for your assignment using the following naming convention:

  The zip file should be called *a#_StudentName_StudentID*, where # is the number of the assignment and *StudentName/StudentID* is your name and ID number respectively. Use your "official" name only - no abbreviations or nicknames; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. For example, for the first assignment, student 12345678 would submit a zip file named like: *a1_Mike-Simon_123456.zip.* if working in a group, the name should look like: *a1_Mike-Simon_12345678-AND-Linda-Jackson_98765432.zip.*

- If working in a team, only one of the members can upload the assignment. Do **NOT** upload the file for each of the members!

⇨ Important: Following your submission, a demo is required (please refer to the course outline for full details). The marker will inform you about the demo times. Please notice that failing to demo your assignment will result in zero mark regardless of your submission.

Evaluation Criteria

Part 1 (6 points)	
UML representation of class hierarchy & **JavaDoc** documentations	2 pt
Proper use of packages	1 pt
Correct implementation of the classes	1 pt
Constructors, toString() & equals()	1 pt
Driver program & general correctness of code	1 pt
Part 2 (4 points)	
Access rights	2 pt
copyCityBuss() and explanantion of its behaviour	2 pt
Total	**10 pts**