Concordia University
Comp 249 – Fall 2023
Object-Oriented Programming II
Assignment 1
Due 11:59 PM – Sunday, October 8, 2023

⇨ **IMPORTANT NOTE:** IN THIS ASSIGNMENT YOU ARE NOT PERMITTED TO USE ANY OF THE JAVA BUILT-IN CLASSES, SUCH AS ArrayLists, Hash Maps, etc. Using any of these will result in zero marks! In other words, you need to code whatever is needed by yourself!!!

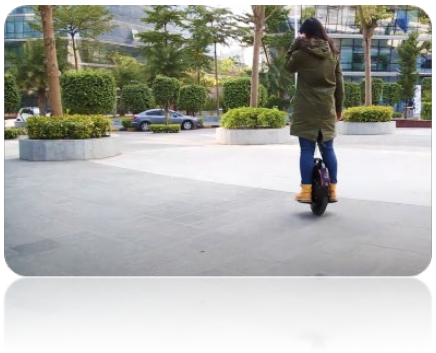
Purpose: The purpose of this assignment is to practice class inheritance, and other Object-Oriented Programming concepts such as constructors, access rights, method overriding, etc. The assignment would also allow you to practice the notion of package creation. This assignment contains two parts. You need to complete Part I to be able to do Part II. You should however handle each part as if it was a separate assignment (i.e. you should not modify Part I after finishing with it; rather create a copy of that part and use it to create Part II. This is needed so that you can finally submit both parts!

The Classes!

Various classes, possibly for transportation or otherwise, are described as follows:

- A **WheeledTransportation** class with the following: *number of wheels* (int type), *maximum speed* (double type), and a serial number (long type).

- A **Train** is a **WheeledTransportation** that in addition it has the following: *number of vehicles* (int type) and *the name of the starting station* (String type), and the *name of the destination station* (String type), such as Montreal, New York, etc. A Train object must have a unique serial number that is assigned automatically upon creation. The initial serial number for any of these Train objects starts at 10000.



- A **Metro** is a **Train** that in addition it has the following: *total number of stops* (int type) which indicates the total number of stations between, and including, the starting and destination stations. A Metro object must have a unique serial number that is assigned automatically upon creation. The initial serial number for any of these Metro objects starts at 25000.

- A **Tram** is a **Metro** that in addition has the following: *year of creation* (int type), which indicates the year the tram was created. A Tram objects must have a unique serial number that is assigned automatically upon creation. The initial serial number for any of these Tram objects starts at 30000.

- A **Monowheel** is a **WheeledTransportation** that in addition it has the following: m*aximum weight* (double type) which indicates the maximum weight the Monowheel supports. A Monowheel object must have a unique serial number that is assigned automatically upon creation. The initial serial number for any of these Monowheel objects starts at 55000.

- A **Ferry** class has the following attributes: *maximum speed* (double type), and *maximum load* (double type), which indicates the maximum load that the ferry can array. A Ferry object must have a unique serial number that is assigned automatically upon creation. The initial serial number for any of these Ferry objects starts at 70000.

- An **Aircraft** class has the following attributes: *price* (double type), m*aximum elevation* (double type) which indicates the maximum possible hight the aircraft can reach, and a serial number (long type).

- A **World War II Airplane** is an Aircraft that in addition it has the following attribute: twin engine (boolean type), which indicates whether or not the plane has a twin engine. A World War II Airplane object must have a unique serial number that is assigned automatically upon creation. The initial serial number for any of these objects starts at 80000.

** Image Sources:
https://lisbonlisboaportugal.com
https://www.youtube.com
https://www.nst.com.my
https://www.bahn.com
https://www.popsci.com
https://www.theatlantic.com

Part I:

1. Draw a single UML representation for the hierarchy of the above-mentioned classes. 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). In case additional classes are needed to reflect how the above classes are related in real-life, you can add these classes.

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

   - You must have 6 different Java packages for the classes. The first package will include the **WheeledTransportation** class. The second package will include the **Train**, and the **Tram** classes. The third package will include the **Metro** class. The fourth package will include **Monowheel** class. The fifth package will include the **Aircraft** and the **World War II Airoplane** classes, and the last package will include the **Ferry** class.

   - For each of the classes, you must have at least three constructors, a default constructor, a parameterized constructor (which will accept enough parameters to initialize ALL the attributes of the created object from this class) and a copy constructor. For instance, if any of these classes has 7 attributes (including the serial number), then the parameterized constructor must accept 6 parameters to initialize all its 6 attributes – the serial number is set automatically. The copy constructor creates a new object that is an exact copy of the passed object, with the exception of the serial number.

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

   - For each of the classes, where objects are created with concrete serial numbers, you must have a method called ***getNextSerialNumber()***. This method will indicate the next serial number that will be given to the next created object from these classes.

   - 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 clear description and information of the object (i.e *"This Train – serial #10204 - has 60 wheels, has a maximum speed of 140 km/hr. It has 6 vehicles and its starting and destination stations are Montreal and Toronto"*).

     – The ***equals()*** method must first verify 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 is conducted to see if the two objects are actually equal. Two objects are equal if the values of all their attributes, of course with the exception of the serial number, are equal.

   - For all classes you **must** use the appropriate access rights, which allow most ease of use/access **without compromising security**. Do not use most restrictive rights unless they make sense!

   - 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 calling a public method from that base class to access the attribute.

3. Write a driver program (that contains the main() method), which will utilize all of your classes, and trigger all the methods you written. You should always keep in mind that if a code is written but never get

triggerred and tested, then it may not work correctly when needed; so test all your code! The driver class can be in a separate package or in any of the already existing packages. Besides the main() method, the driver will also include another static method called ***findLeastAndMostExpensiveAircraft().*** The description this is as follows:

⇨ The ***findLeastAndMostExpensiveAircraft*** **()** should be able to accept an array that contains **mixed** objects from any of the 8 classes described above, as a parameter. The method however must find both the least-expensive and the most-expensive Aircraft objects in that array (if any exists). If so, the method must find, then display (using *toString*), the information of these two Aircraft objects. If only one Aircraft object exists in the passed array, then it is considered as the least and the most expensive. If no Aircraft objects are found, the method must display something to that effect.

1. In the **main()** method you must:
   - Create at least 16 objects from the 8 classes, and display their information (you must take advantage of the ***toString()*** method).
   - Test the equality of some of the created objects using the ***equals()*** method. You should test at least the equality of two objects from different classes, two objects from the same class with different values and two objects with similar values. In other words, you should include enough test cases to test your implementation.
   - Create two arrays, *each* of 15 to 20 of these mixed objects and fill these arrays with various objects from these classes. The first array must include at least one object from each of the classes; while the second array should not have any Aircraft objects.

     (**HINT**: Again, do you need to add something else to the classes described above? If so; go ahead with that!)
   - Finally call the ***findLeastAndMostExpensiveAircraft()*** once with the first array as a parameter and once with the second array. This should display what is needed!

5. **Does my program work correctly, or does it misbehave! WHY?**

Investigate the output of your program! You will need to submit a separate document (pdf, MS-Word, or text) along with your assignment indicating **whether or not the resulted display from** the ***findLeastAndMostExpensiveAircraft()*** **is correct. In either case, you need to explain the reason for the correctness or failure of your program's behavior/output.**

Part II

In that part, you need to modify/expand the implementation from Part I as follows:

1. Create a new driver program (again, keep in mind that you should treat that as if it was a different assignment. You can however copy/reuse any code from Part I). Besides the main() method, you need to add another static method (add it above the main() method), called ***copyTheObjects***. The method will take as input an array of these objects (the array can be of any size and should have mixed objects from all of the classes) and returns a copy of that array. That is to say, the method needs to create an array of the same length as the passed array, copy all objects from the passed array to a new array, then return the new array. Your copy of the objects **will automatically** depend on the **copy constructors** of the different listed classes.

2. You **must** consider the following restrictions: **Do NOT attempt to explicitly find the exact type of the objects being copied, do NOT attempt to find the object type inside the copy constructors and Do NOT use clone().**

3. In the driver program, create an array of 15 to 20 objects (must have at least one from each of the classes), then call the *copyTheObjects()* method to create a copy of that array.

4. **Does my program work correctly, or does it misbehave! WHY?**

Display the contents of both arrays, then submit a separate document (pdf, MS-Word, or text) along with your assignment indicating **whether or not the copying is correct. In either case, you need to explain the reason for the correctness or failure of your program's behavior/output.**

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). Groups of more than 2 students = zero mark for all members! Submit only ONE version of an assignment. If more than one version is submitted the first one will be graded and all others will be disregarded.
- Students will have to submit their assignments (one copy per group) using Moodle. Assignments must be submitted in the right submission folder of the assignments. **Assignments uploaded to an incorrect folder will not be marked and result in a zero mark. No resubmissions will be allowed.**
- Naming convention for zip file: Create one zip file, containing all source files and produced documentations 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 (Please read very carefully): Additionally, which is very important, a demo will take place with the markers afterwards. Markers will inform you about the details of demo time and how to book a time slot for your demo. If working in a group, both members must be present during demo time. Different marks may be assigned to teammates based on this demo.

- **If you fail to demo, a zero mark is assigned regardless of your submission.**
- **If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a <u>penalty of 50% will be applied.</u>**
- **Failing to demo at the second appointment will result in zero marks and <u>no more chances will be given under any conditions.</u>**

<u>Evaluation Criteria</u>

Part I	
UML representation of class hierarchy	1.5 pts
Proper use of packages	1 pt
Correct implementation of the classes	1 pt
Constructors & toString() & equals()	1 pt
findLeastAndMostExpensiveAircraft() & correctness/analysis of the code	2 pts
Part II	
copyTheObjects() and its behaviour	2.5 pts
Driver program & general correctness/analysis of code	1 pts
Total	**10 pts**