Write an application that will be used to simulate and maintain the logistics of the railway infrastructure, including railway stations, connections, line intersections and different train implementations.

You should create all the necessary classes and supplement each class with at least two additional fields and methods that correspond to the class (not getters and setters).

As part of the application, we must be able to create a new locomotive, railroad cars, railway stations and connections between stations from the menu (apart from other functionalities described below). We also need to be able to perform tasks such as attaching a railroad car to a locomotive, loading people/cargo onto railroad cars, etc. Object removal must also be considered.

Each trainset has information about the locomotive and the railroad cars connected to it. Each locomotive is defined by:

- maximum number of railroad cars
- maximum weight of the transported load
- the maximum number of railroad cars that need to be connected to the electricity grid

Each locomotive has basic information about itself (name, home railway station, source and destination railway station) and its unique identification number assigned automatically when creating an object.

In addition, locomotives have a speed assigned to them. This speed will decrease or increase (randomly) by 3% every 1 second. Based on this speed, the movement of trainsets along the routes should be implemented.

There are different types of railroad cars. Each type of railroad car has different characteristics (e.g. shipper, security information, net weight, gross weight, number of seats, etc.). For each type of railroad car, you must add at least two unique features to those listed. Each railroad car has its unique identification number assigned automatically when creating an object.

We have at our disposal, among others:

- passenger railroad car that requires connection to the locomotive's electrical grid
- railroad post office that requires connection to the locomotive's electrical grid
- railroad baggage and mail car
- railroad restaurant car that requires connection to the locomotive's electrical grid
- basic railroad freight car
- heavy railroad freight car
- refrigerated railroad car, which is a type of basic freight railroad car that requires connection to the locomotive's electrical grid
- railroad car for liquid materials, which is a kind of basic freight railroad car
- railroad car for gaseous materials, which is a kind of basic railroad freight car
- railroad car for explosives, which is a kind of heavy railroad freight car
- railroad car for toxic materials, which is a kind of heavy railroad freight car
- railroad car for liquid, toxic material, which is a kind of heavy railroad freight car, and has the characteristics of a railroad car for liquid materials

In trainsets, we are limited not only by the maximum number of railroad cars but also by the maximum weight of the transported load, therefore, before connecting another railroad car, we must check whether the next railroad car will not exceed these limits. If adding another railroad car is impossible, an appropriate message will be displayed based on the raised exception.

Trainsets move along routes (determined according to the indicated objects of the starting and destination railway station). The route between the railway stations must be determined each time (it does not have to be the shortest route, but each time it must be determined algorithmically on the basis of the graph of railway connections).

Trainset stops at every station it encounters and waits 2 seconds. After reaching the destination station, the trainset waits for 30 seconds, and then the train starts its return journey (generates a new route) and runs all the time in this cycle.

Collision prevention should also be implemented, in which a maximum of one trainset can move between two stations. If there is a situation where there is already a train on the route, the others wait in the queue for the route to be released, and in the order of appearance, they will be passed through the desired traction between railway stations.

All things related to time should be done using the Thread class (the *Timer* class is not allowed). Careful and correct synchronization of all threads must be ensured. You cannot combine different functionalities into one thread.

When the trainset exceeds the speed of 200km/h, the application must inform the user with an appropriate message on the console in the form of an exception of the type ***RailroadHazard***. This exception contains basic trainset information provided by the exception message.

We need to be able to display (after providing the trainset identifier) a report containing:

- all basic information about trainset
- % of the distance completed between the starting and destination railway stations
- a summary of information about railroad cars, the number of people based on the goods transported
- % of the distance completed between the nearest railway stations on your route.

The information should be displayed on the console after selecting the appropriate option. In addition, a list of all existing trainsets should be automatically added to the *AppState.txt* file every 5 seconds (in text, user-readable form). The information should be saved following the rules below:

- railroad cars in trainsets sorted in ascending order by weight.
- trainsets sorted in descending order of route length between the start and end of the route.

All exceptions should be handled and communicated to the user with appropriate messages without the need to interrupt and restart the application.

Prepare (generate) at least 100 railway stations with connections between them, 25 trainsets (with randomly 5-10 railroad cars each).

In addition to the main flow of the application, create a separate file (*Presentation.java*) with an additional method *main* containing examples simply showing each of the functionalities.

The project is based on GUI material.

Attention:

- ***In the case of receiving a project with significant deficiencies in implementation or a non-compiling solution, the result for such a project will be 0 points.***
- ***Lack of knowledge of any line of code or plagiarism will result in obtaining 0 points for this project with the possibility of failing the entire subject.***
- ***Not only the practical and substantive correctness of the solution will be assessed, but also the optimality, quality and readability of the code written by you.***
- ***An important part of the project is the use of: inheritance, collections, interfaces or abstract classes, lambda expressions, Java Generics, additional functionalities or structures and other characteristic elements presented in the classes and lecture.***