# Software Engineering and Programming Basics - WS2024/25
# Instructions for the Mandatory Assignment

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Professorship of Software Engineering

October 30, 2024

## General Instructions

The assignment is a mandatory task that you need to complete over the course of the semester. Please read all the instructions carefully. Submissions that do not adhere to the instructions will be graded with 5.0.

- You need to create a Java-Project which contains three packages with the following names:
  - interfaces
  - classes
  - tests
- Download the *Material_for_Students.zip* folder from OPAL to use in your project. You are not allowed to make any changes to the interfaces or the class names.
- You are generally allowed to make additions, such as additional classes and methods which are not specified within this document. Just note that additional methods should not be added to the interfaces.
- Add any additional classes which are required to run your program to the package *classes*.
- You are allowed to use classes and methods from the Java standard library that are introduced in the Bonus Lecture Video "Important Java Classes". These are the classes `java.lang.Object`, `java.lang.String`, `java.lang.StringBuilder`, `java.lang.Math,` and `java.util.Arrays`. You are *not* allowed to import any other classes.
  - Expection: You are allowed to import additional classes for testing purposes within the package *tests*.
- You are *not* allowed to use any third party libraries!
- Your program needs to pass the given unit tests from the *tests* package provided in the *Material_for_Students.zip* folder without fail. A few more notes on these tests:
  - These tests also function as examples for the required functionality of methods, thus you should not make any changes to them! You can, however, add more tests of your own, including modified copies of the provided unit tests.
  - We recommend that you add the tests to your project after you have created all the required classes, to avoid your IDE showing a lot of errors.
  - You need to add `JUnit5` to your project in order for the tests to work. Please follow the instructions of your IDE.
  - If you have any troubles with getting the tests to run, please do not hesitate to contact the instructors for help.
- Since this is an examination, you need to solve the task on your own. You can talk to other people about it, however it is not allowed that someone else writes the program for you or that you copy code from each other. Each submission will be checked for plagiarism! If plagiarism is found, all of the concerned submissions will be graded with 5.0, regardless of which one was the original!
- You are allowed to use code snippets that you find online. However, to avoid being flagged for plagiarism (in case someone else finds and uses the same code snippet), you need to clearly indicate which parts of your code have been copied by stating the source in a comment in the code.

- You are also allowed to use AI-based tools such as ChatGPT. Like with code from other sources, you need to clearly indicate any part of your code which was created with the assistance of such tools, by stating it in comments in the code.
- You need to submit your code as .java files. No other type of file will be accepted.
- You need to submit your complete classes package. You can also submit your tests package, but it is not mandatory.
- In addition to your source code, you will need to submit a short documentation. Details follow below.
- Submit your Source Code and your Documentation in OPAL in the "Assignment Submission" course node.
- Deadline for submissions is February 14th, 2025.

If you have any questions, please post them in the corresponding thread in the OPAL-Forum! Since this is an examination, it is important that all students have access to all information. If you have a question, it is very likely that someone else has the same question as well, thus it benefits everyone if you post your question directly in the forum! Additionally, please also check the forum first if maybe your question has already been answered there.

## Documentation

You need to submit a documentation on your code. It should be in a suitable format (.pdf) and at maximum 5 pages (or 2500 Words). Its contents should be the following:

- A rough description of how the classes interact with each other (maximum 1 page or 500 words). Include a UML class diagram to support your explanation.
- Explain for each of the required classes which attributes you used and why.
- A short explanation for each additional method you created that was not given by the interfaces. Explain in 1 or 2 sentences what the method does/is needed for.
- A short explanation of how your implementations of the following methods work:
  - List.swap
  - EmergencyDispatchCenter.sortVehicles
  - EmergencyDispatchCenter.dispatchVehicles

## Grading

### Minimal requirements to pass

- Your submission needs to contain all required classes and the required documentation.
- The program needs to compile and run.
- The program needs to pass the basic Unit Tests that are provided in the *Material_for_Students.zip* folder in OPAL.
- The program needs to fully implement the given interfaces and there must not be any changes to the interfaces.
- The program needs to pass the plagiarism check.

**If any one of these points is not fulfilled, the submission will be graded with 5.0!**

### Further grading criteria

- Functionality (60%)

  Including edge cases that are not covered by the provided unit tests.
- Clean Code (30%)
  - Use appropriately describing identifier names for variables, methods and classes.
  - Keep your methods and classes a sensible size.
  - Avoid code duplication.
  - Make sensible use of comments.
  - In general: Write code that is easily readable.
- Documentation (10%)

## Task Description

Your task is to program a simulation of an emergency dispatch center.

### User Stories

An emergency dispatch center is a place where emergency calls are answered and help is planned and sent according to the emergency situation. An emergency dispatch center has access to a set number of fire stations. For the sake of simplicity, a fire station has fire trucks and ambulances stored until they are needed. An ambulance can have a doctor present or not, which determines which kind of patients it can handle. Your task is to simulate an emergency dispatch center in its working. We have formulated some user stories to help you get an idea of what the processes in the emergency dispatch center look like, that your code needs to represent:

- When a new emergency call is made, it needs to be registered in the waiting list.
- Once an emergency call is made, the vehicles needed for handling the emergency are planned and dispatched. The emergency will be handled, which takes a certain amount of time.
- Large emergencies will have priority over middle or small sized cases. A medical emergency is a middle sized emergency.

Every vehicle registered in an emergency dispatch center has a unique identifier, which looks like this:
[station number]/[vehicletype]/[vehicle number]

There are the following vehicle types:

- 11 - command vehicle
- 33 - ladder truck
- 49 - fire engine
- 52 - rescue truck
- 78 - hazmat truck
- 81 - ambulance with doctor
- 83 - ambulance without doctor

The vehicle number tells the number of vehicles of the same type stationed on the same station.
Example: If station 1 has three fire engines stationed in it, they would be called: 1/49/1, 1/49/2, and 1/49/3. But one fire engine and one ladder truck would both have vehicle number one. So the ladder truck from station 1 would have the ID 1/33/1, whereas the first fire engine of station 1 has the ID 1/49/1.

### Required Classes

- Ambulance
- Emergency
- EmergencyDispatchCenter
- FireStation

- FireTruck
- List
- Node

  and the following enums
- FireTruckKinds which can take the following values:
  - FireEngine, LadderTruck, HazmatTruck, RescueTruck, CommandVehicle
- EmergencyKinds which can take the following values:
  - FireLarge, TechnicalEmergencyLarge, HazmatEmergencyLarge, FireMiddle, TechnicalEmergencyMiddle, HazmatEmergencyMiddle, FireSmall, TechnicalEmergencySmall, HazmatEmergencySmall, MedicalEmergency

## Required vehicles at an emergency

Each emergency requires different vehicles to respond. Here is listed which vehicles are required at each emergency kind.

- FireSmall: 1x fire engine
- TechnicalEmergencySmall: 1x fire engine, 1x ladder truck
- HazmatEmergencySmall: 2x fire engine
- FireMiddle: 3x fire engine, 1x ladder truck, 1x command vehicle
- TechnicalEmergencyMiddle: 1x rescue truck, 2x fire engine, 1x command vehicle
- HazmatEmergencyMiddle: 1x hazmat truck, 1x rescue truck, 2x fire engine, 1x command vehicle
- FireLarge: 1x rescue truck, 5x fire engine, 2x ladder truck, 1x command vehicle
- TechnicalEmergencyLarge: 2x rescue truck, 4x fire engine, 1x ladder truck, 1x command vehicle
- HazmatEmergencyLarge: 2x hazmat truck, 1x rescue truck, 4x fire engine, 1x ladder truck, 1x command vehicle
- MedicalEmergency: no fire trucks needed

In addition to the fire trucks, ambulances are needed to care for injured people. For each injured person, one ambulance is required. For each injured person who needs to be treated by a doctor, an ambulance with a doctor is required.

In the following, we explain in detail the requirements for each of the required classes and methods:

## Emergency

The class `Emergency` implements the interface `I_Emergency`. It contains the following methods:

```
public Emergency (String location, EmergencyKinds kind, int patients, int
    patientsDoc)
```
This constructor creates an object of the class Emergency.

```
public String getLocation()
```
Returns the emergency's location.

```
public void setLocation(String location)
```
Sets the emergency's location.

```
public EmergencyKinds getKind()
```
Returns the kind of emergency.

```
public void setKind(EmergencyKinds kind)
```
Sets the kind of emergency.

```
public int getCasualties()
```
Returns the amount of people injured in the emergency.

```
public void setCasualties(int casualties)
```
Sets the amount of people injured in the emergency.

```
public int getCasualtiesNeedsDoctor()
```
Returns the amount of injured people that need a doctor in the emergency.

```
public void setCasualtiesNeedsDoctor(int casualtiesNeedsDoctor)
```
Sets the amount of injured people that need a doctor in the emergency.

## Ambulance

The class `Ambulance` implements the interface `I_Ambulance`. It contains the following methods:

```
public Ambulance (int stationNum, int vehicleNum, boolean hasDoctor)
```
This constructor creates an object of the class Ambulance with its unique identifier (ID), which is created in the way described above.

```
public String getID()
```
Returns the ID of the ambulance.

```
void setID(String id)
```
Sets the ID of the ambulance.

```
public boolean getHasDoctor()
```
Returns whether the ambulance has a doctor.

```
public void setHasDoctor(boolean hasDoctor)
```
Sets whether the ambulance has a doctor.

## FireTruck

The class `FireTruck` implements the interface `I_FireTruck`. It contains the following methods:

`public FireTruck (int stationNum, int vehicleNum, FireTruckKind kind)`
This constructor creates an object of the class FireTruck with its unique identifier (ID), which is created in the way described above.

`public String getID()`
Returns the ID of the fire truck.

`void setID(String ID)`
Sets the ID of the fire truck.

`public FireTruckKinds getKind()`
Returns the kind of fire truck.

`public void setKind(FireTruckKinds kind)`
Sets the kind of fire truck.

## Node

The class `Node` implements the interface `I_Node` and represents a generic Node class. It contains the following methods:

`public Node ()`
This constructor creates an empty object of the class Node.

`public Node (T content)`
This constructor creates an object of the class Node with the content of the Node already set.

`public T getContent()`
Returns the content of the Node.

`public void setContent(T content)`
Sets the content of the Node.

`public Node<T> getPrev()`
Return the previous node.

`public void setPrev(Node<T> prev)`
Sets the previous node.

`public Node<T> getNext()`
Return the next node.

`public void setNext(Node<T> next)`
Sets the next node.

## List

Which is a generic double linked list class and implements the interface `I_List`. The first element in the list (= the list's head) has index 0. It implements the following methods:

`public List ()`
This constructor creates an empty object of the class List.

`public List (Node<T> head)`
This constructor creates an object of the class List with a Node.

`public Node<T> getHead ()`
Returns the head of the list.

`public Node<T> getTail ()`
Returns the tail of the list.

`public int getSize ()`
Returns the number of nodes in the list.

`public boolean append (Node<T> node)`
Adds a node at the end of the list. If successful, it returns true, else it returns false.

`public boolean insert (int index, Node<T> node)`
Inserts a node at the given index of the list. If the given index is 5, the new node should be at index 5 and the node formerly at the index 5 is now at the index 6. If successful, it returns true, else it returns false.

`public boolean remove (int index)`
Removes the node at the given index. If successful, it returns true, else it returns false.

`public boolean remove (T content)`
Removes the first node in the list with the given content. If successful, it returns true, else it returns false.

`public Node<T> get (int index)`
Returns the node at the given index. If successful, it returns the node, else it returns null.

`public boolean swap (int indexOne, int indexTwo)`
Swaps the nodes at the given indexes. Do not just switch the content of the nodes, but the exact nodes. If successful, it returns true, else it returns false.

## FireStation

The class `FireStation` implements the interface `I_FireStation`. It contains the following methods:

`public FireStation (int number, String name)`
This constructor creates an object of the class FireStation.

`public int getNumber()`
Returns the number of the fire station.

`public void setNumber(int number)`
Sets the number of the fire station.

`public String getDistrict()`
Returns the district of the fire station.

`public void setDistrict(String district)`
Sets the district of the fire station.

`public List<FireTruck> getFireTrucks()`
Returns a list of the fire trucks of the fire station.

`public void setFireTrucks(List<FireTruck> fireTrucks)`
Sets a list of the fire trucks of the fire station.

`public List<Ambulance> getAmbulances()`
Returns a list of the ambulances of the fire station.

`public void setAmbulances(List<Ambulance> ambulances)`
Sets a list of the ambulances of the fire station.

`public boolean addVehicle (FireTruck truck)`
Adds the given FireTruck object to the list of fire trucks of the fire station. If successful, it returns true, else it returns false.

`public boolean addVehicle (FireTruckKinds kind)`
Adds a new fire truck of the given kind to the list of fire trucks of the fire station. If successful, it returns true, else it returns false.

`public boolean addVehicle (Ambulance ambulance)`
Adds the given Ambulance object to the list of ambulances of the fire station. If successful, it returns true, else it returns false.

`public boolean addVehicle (boolean hasDoctor)`
Adds a new ambulance to the list of ambulances of the fire station. If successful, it returns true, else it returns false.

`public boolean removeVehicle (FireTruck truck)`
Removes the given fire truck from the list of fire trucks of the fire station. If there are vehicles with a higher ID, make sure that there are no gaps between the vehicle-ids. If successful it returns true, else it returns false.

`public boolean removeVehicle (Ambulance ambulance)`
Removes the given ambulance from the list of ambulances of the fire station. If there are vehicles with a higher ID, make sure that there are no gaps between the vehicle-ids. If successful, it returns true, else it returns false.

`public boolean removeVehicle (String ID)`
Removes the vehicle with the given ID. If there are vehicles with a higher ID, make sure that there are no gaps between the vehicle-ids. If successful, it returns true, else it returns false.

`public void printVehicles ()`
Prints all vehicles in this station in ascending order by ID according to the format:
”station [stationnumber]:
ID - vehicleKind
...”

`public void sortVehicles ();`
Sorts all vehicles in this station ascending by their ID.

## EmergencyDispatchCenter

```
public EmergencyDispatchCenter ()
```
This constructor creates an object of the class EmergencyDispatchCenter.

```
public List<FireStation> getFireStations()
```
Returns the list of fire stations.

```
public void setFireStations(List<FireStation> fireStations)
```
Sets the list of fire stations.

```
public List<Emergency> getEmergencies()
```
Returns the list of emergencies.

```
public void setEmergencies(List<Emergency> emergencies)
```
Sets the lists of emergencies.

```
public boolean addFireStation (FireStation station)
```
Adds a fire station which the dispatch center can use in case of an emergency.

```
public boolean addVehicle (int station, boolean hasDoctor);
```
Adds an ambulance to the given station.

```
public boolean addVehicle (int station, FireTruckKinds kind);
```
Adds a fire truck of the given kind to the given station.

```
public boolean removeVehicle (String ID);
```
Removes the vehicle with the given ID from the according fire station.

```
public Emergency newCall (String location, EmergencyKinds kind, int patients,
    int patientsNeedDoctor);
```
Creates an emergency call with the given attributes.

```
public boolean addCalltoList (Emergency call);
```
Adds the given emergency call to the list of emergencies.

```
public void sortCalls ();
```
Sorts the emergency calls in the list of emergencies from large to small. If there are multiple calls of the same size, their order is not relevant.

```
public void sortVehicles ();
```
Sorts the emergency vehicles in ascending order by ID.

```
public boolean dispatchVehicles ();
```
Dispatches fire trucks and ambulances according to the emergency kind and casualty number. Large emergencies should be attended to first, then middle-sized emergencies, and finally small emergencies. If all necessary vehicles can be dispatched to all calls in the list of emergencies, return true, else return false.

```
public void printRespondingVehicles (Emergency call);
```
Prints all vehicles responding to the given call in ascending order by their ID in the format:

```
"To $EmergencyType in $Location is responding:\n$verhicleID - $vehicleKind"
```

Variables that are marked with $ need to be replaced with the concrete values of the case.
Repeat

```
\n$verhicleID - $vehicleKind
```

for each vehicle that is dispatched to the emergency. An ambulance with doctor is printed as "$ID - Ambulance(Doctor)", an ambulance without a doctor is printed as "$ID - Ambulance".
Important note: Use the System.out.print() function, *not* the System.out.println() function for the corresponding unit test to work. (If you do not get the test to run at all, please contact the instructors.)