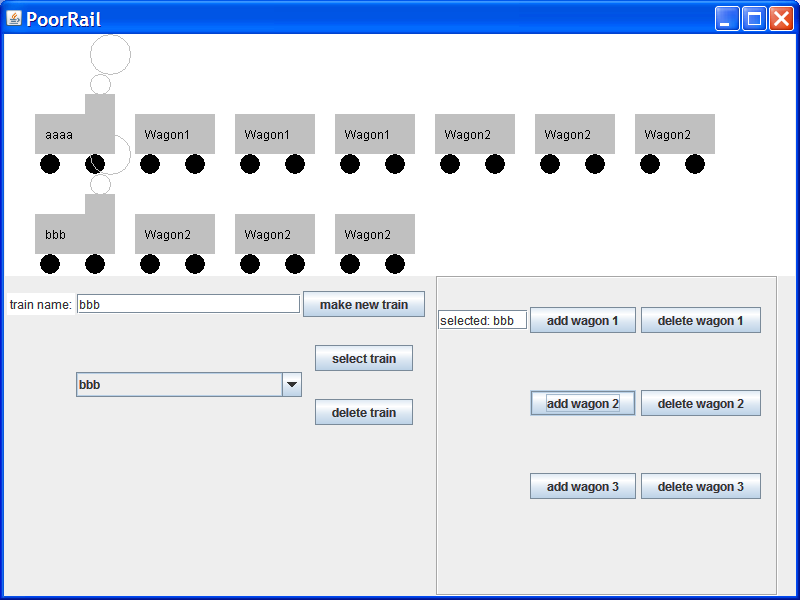
**Patterns & Frameworks - Assignment 2 - From PoorRail to RichRail**

A public transport company has a little system with which they are administrating their trains and wagons. The system has been very poorly designed and constructed. The code is not maintainable and full of errors. A redesign is required because new requirements have to be added. Our goal is not to improve the current code but to start from scratch with a good design based on our design principles (SOLID).

For completeness the code that implements the current system is available in eclipse project “poor-rail”. A screenshot can be seen below.



At the time the system was built they only had 3 different types of wagons, but now they want to be able to add more wagon types. They also want the possibility of displaying their trains in different ways. To be more flexible in using the program it was decided to introduce a command line interface and a domain-specific language (DSL). Finally it should be fairly easy to extend the way of output details like for example logging.

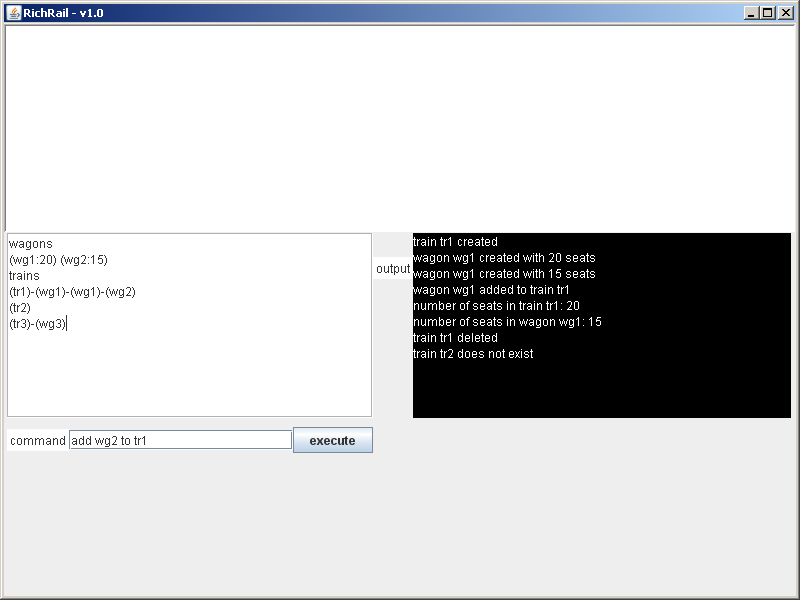
To summarize the new design and new prototype should include the current requirements (create, select, delete trains, add wagons and delete wagons).

Next to that the following requirements have to be included in the new software:

* The display of the existing trains including wagons and wagon types has to be interchangeable, meaning that also other display types could be easily added. It also has to be possible to show one of the existing displays twice or more (in another window). The realization of this requirement has to be shown in the final presentation
* the output has to be extensible (in such a way that it also can be e.g. logged to a file)
* no persistency of the data is required
* command line interface based on DSL (explained in next section)

**Command line interface based on DSL**

Experienced users prefer a command line interface to construct the train settings. Commands are based on a grammar and introduce a domain specific language (DSL). The next screenshot shows commands in the command line interface (e.g.: add wg2 to tr1), the display of the train settings (wagons …….) and a logging screen (train tr1 created). The grammar that defines the possible commands is described in the next section with some examples.



## Grammar

This grammar describes the Domein-specific language using a modified version of the Backus-Naur form (http://en.wikipedia.org/wiki/Backus%E2%80%93Naur\_Form).

Responses are displayed in extra textfield. If command is not correct, an error-message is displayed: “command not correct”. Default amount of seats is 20. Also show list with available wagons.

grammar RichRail;

command : newcommand | addcommand | getcommand | delcommand | remcommand;

newcommand : newtraincommand | newwagoncommand;

newtraincommand : 'new' 'train' ID;

newwagoncommand : 'new' 'wagon' ID ('numseats' NUMBER)?;

addcommand : 'add' ID 'to' ID;

getcommand : 'getnumseats' type ID;

delcommand : 'delete' type ID;

remcommand : 'remove' ID 'from' ID;

type : ('train') | ('wagon');

ID : ('a'..'z')('a'..'z'|'0'..'9')\*;

NUMBER : ('0'..'9')+;

WHITESPACE : ( '\t' | ' ' | '\r' | '\n'| '\u000C' )+;

**Examples:**

language-words are **bold**

identifiers are *italic*

numbers are normal

**new train** *tr1*; // response is “train tr1 created”

**new wagon** *wg1*; // response is “wagon wg1 created with 20 seats”

**new wagon** *wg2* **numseats** 15; // response is “wagon wg2 created with 15 seats”

**add** *wg1* **to** *tr1*; // response: “wagon wg1 added to train tr1”

**getnumseats train** *tr1*; // response: “number of seats in train tr1: 20”

**getnumseats wagon** *wg2*; // response: “number of seats in wagon wg2: 15”

**delete train** *tr1*; // response: “train tr1 deleted”

**delete train** *tr2*; // response: “train tr2 does not exist”

**remove** *wg1* **from** *tr1*; // response: “wagon wg1 removed from train tr1”

## Organization

This assignment has to be done by student groups of 3 students. Download the source code of **PoorRail** and use this as to get an impression of poor code and possibly an impression of the functionalities. Based on the old and new requirements design and implement the new program **RichRail**. Start with the design based on principles and possibly patterns. Present your initial design (see: studentguide) and gather feedback. Refine your design and implement accordingly in Java. Finally deliver design and code (see: studentguide). Design should contain explanatory text.

## Grading criteria

* Functionality:
  + all required functionality must be implemented:
    - create, select, delete trains, add wagons and delete wagons
    - multiple wagon types
    - multiple displays
    - extensible outputs
    - command line interface through DSL, based on grammar
* Design:
  + Good structured code (components, layers etc.)
  + Principles followed (SOLID)
  + Sensitive and reasonable usage of design patterns
  + Explained
* Coding:
  + Comprehensible, no ‘code smells’
* Not graded:
  + Performance
  + Fancyness

The weight of this assignment is 60%.