Assignment 3

Team number: 41
Team members

Name	Student Nr.	Email
Nikolay Filipov	2691580	n.r.filipov@student.vu.nl
Asim Manzaij	2671198	a.k.manzaij@student.vu.nl
Alejandro Pascual	2708337	a.pascualpintado@student.vu.nl
Andres Latorre	2708338	a.latorremagaz@student.vu.nl

This document has a maximum length of 20 pages (excluding the contents above).

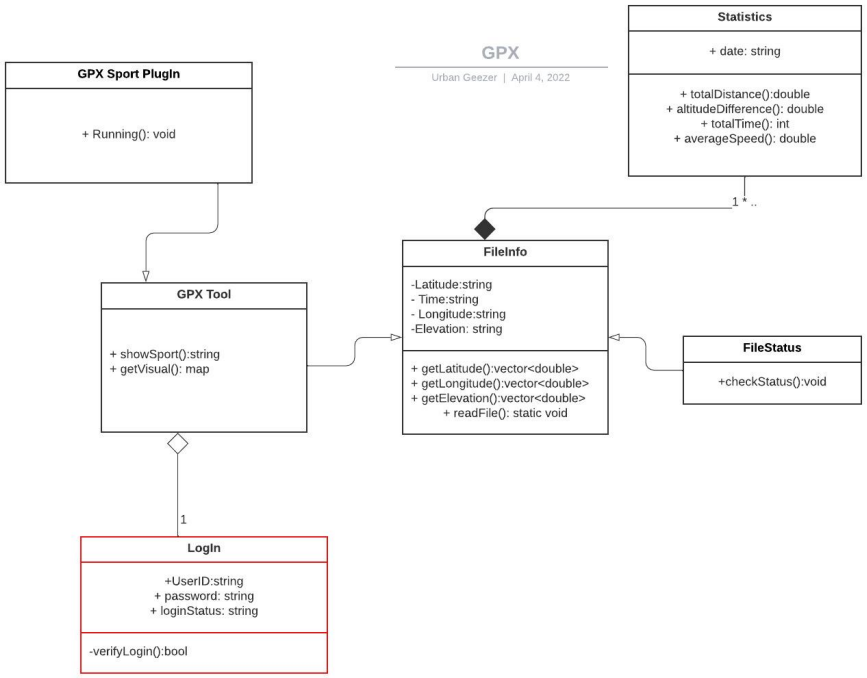
Summary of changes of Assignment 2

Author(s): Nikolay Filipov

- In the first sequence diagram, we improved the name of the loop and changed the dotted lines to have a white arrowhead.
- In the second state machine diagram, we removed the statistics state as that was not itself a state so it shouldn't be on the diagram.
- We removed the calendar from the object diagram because we weren't able to implement it.
- Class diagram was updated with more information and also an API for third-party users.
- Calendar class was removed as it was not implemented in time.
- Added multiplicities in the class diagram where we thought they belonged.
- The state machine diagram about the login case with the calendar was changed to the one with the map as a calendar was not implemented.
- We made note of the feedback to change the profile state to profile display but we were not able to implement it so it was removed.
- Commented code was removed

Application of design patterns

Author(s): Nikolay Filipov



The class diagram with the design pattern in red

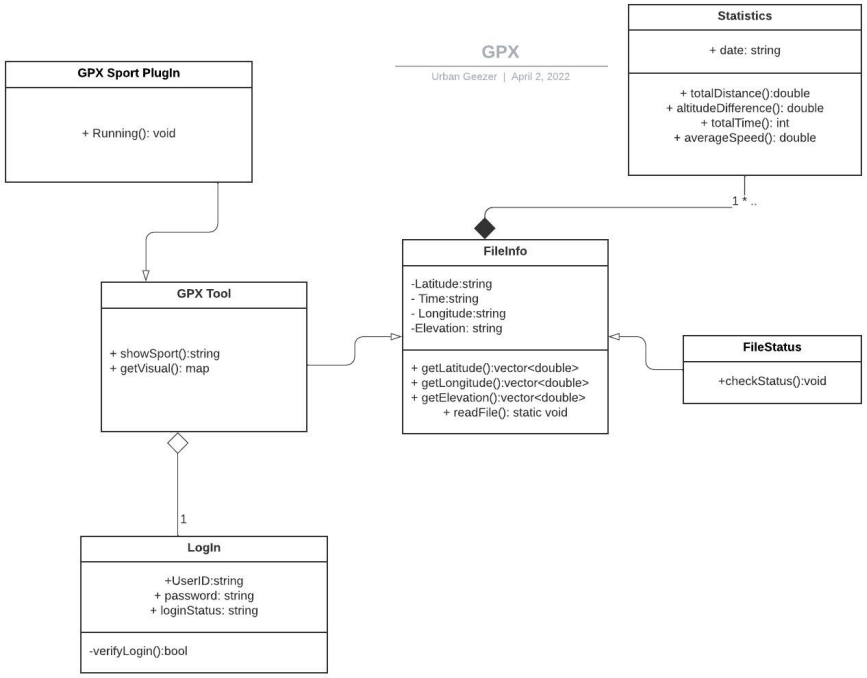
For each application of any design pattern you have to provide a table conforming to the template below.

	DP1
Design pattern	Visitor Design Pattern
Problem	The problem we are trying to solve is one based on security complications. We thought that the map should only be accessible to people with the correct login credentials.
Solution	In our project, we added a login screen in order to add an extra level of security and make the map and file statistics accessible only to people with the correct login credentials. This way we factor out people who are unauthorized to view the map.
Intended use	The intended use of the design pattern is when the user runs the program, they should see a login screen before being able to do anything else. When the login screen pops up the user will be asked to enter a provided username and password. When the login has been successful, only then will the user have the ability to access the map.

Constraints	Currently, the login is only available when looking at the code. It is hardcoded and can be changed only through the code itself by anyone who wishes.
Additional remarks	We weren't kind of confused about what a design pattern was supposed to be. Hopefully this is considered a design pattern.

Class diagram

Author(s): Nikolay Filipov



The main class in our program is the FileInfo which contains the latitude, longitude, time, and elevation as attributes. These three are given as a string and are made public so other classes can use them. The operations of getLatitude, getLongitude, and getElevation() are given as vectors and are made private as they cannot be accessed by other classes. Also, the operation called readFile() is given as a static void and it reads and parses the file.

The next class we have is the GPX Tool which is connected to the GPX File with inheritance as it inherits the information first provided in the GPX File. The GPX Tool then has 2 operations, one of which is getVisual() which shows a map. The other operation is showSport() which is a string and outputs the sport. Both of these operations are made public so they can be accessed by other classes like the user class.

From the GPX Tool there is an aggregation relationship with the next class which is LogIn. Aggregation means that one class is loosely associated with the other. The attributes

of the user class are the userID, the user password, and the login status. All of these attributes are given as strings. The User class also has one operation which is to verify the login status of the user. This is given as a boolean function and inputs true if the user is logged in and false if the user is not logged in. In this example, the user can still login without the GPX Tool necessarily being there. Per GPX Tool , we only have one user who's information will be shown.

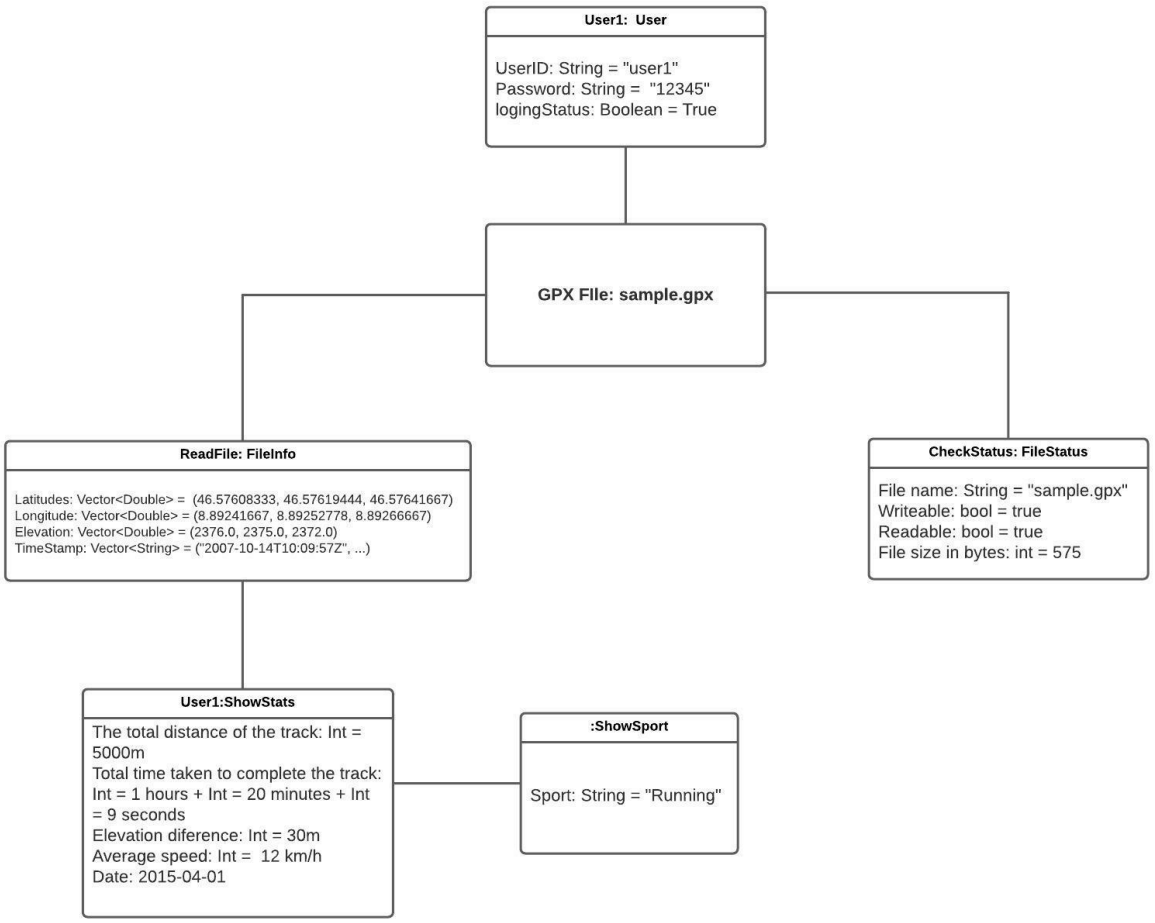
The next class we have is the GPX Sport Plugin which inherits information from the GPX Tool and allows for a third party to create their own sport and input it into the system. It is connected to the GPX Tool class using inheritance. It also has one attribute of its own which in this case is the dates the user logged into the app. The GPX Sport Plugin has only one method for now and that is whether the sport is running, and that is given as a void.

The next class is the statistics class. The class calculates the total distance of the track. It also obtains the total time taken as an integer, the altitude difference as a double and the average speed as a double. It is connected to the GPX file class using composition meaning that if the GPX class is destroyed, the statistics class will also be destroyed. There can be one or more statistics for every file given. Within the statistics class, we have an attribute of date characterized as a string and an operation of totalDistance() also as a double.

The final class is the FileStatus class which has one method. The method is to determine the status of the file and to check whether it is readable. This class is connected to the GPX file class using inheritance.

Object diagram

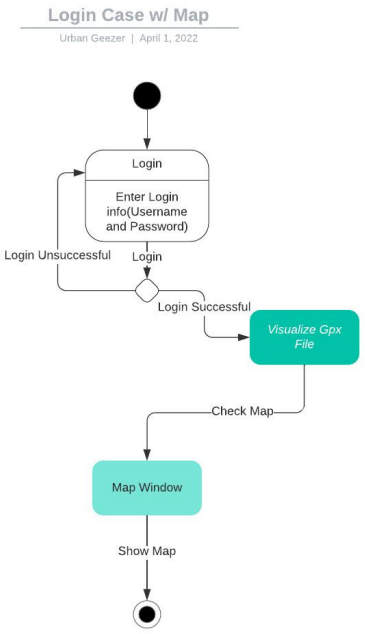
Author(s): Andres, Alejandro



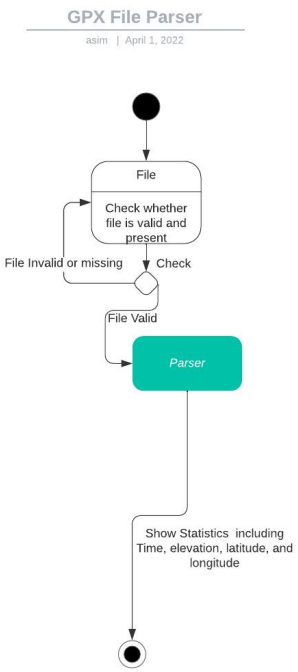
Our object diagram shows a snapshot of how our program would run when tested with a sample gpx file. In this example we can see how the different classes are related. For example, we observe the clear relation between the class Statistics() and the class Showsport() where in correlation to the data obtained we can determine the sport performed by the user. Fileinfo() and FileStatus() obtain their results directly from the sample.gpx file.

State machine diagrams

Author(s): Nikolay Filipov



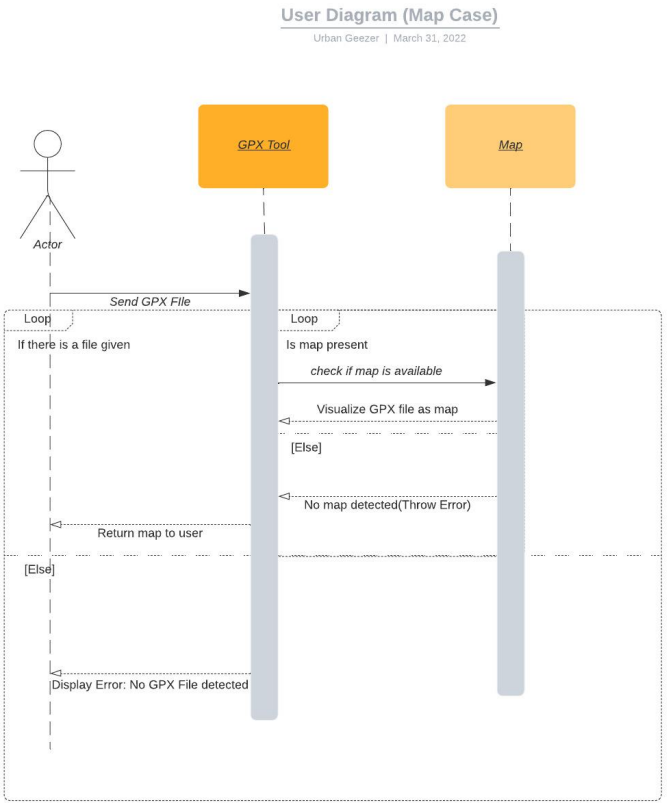
The User activity in order to visualize the GPX file starts with the process of the user login. It is part of the user class. If the user is unable to login, it takes them back to the login screen. If the login is successful, it sends the user to the next state which is the visualize GPX file . From the visualize state, the user can check the map and it sends them to the Map Window state. The system then shows the map and this system of events is then finished.



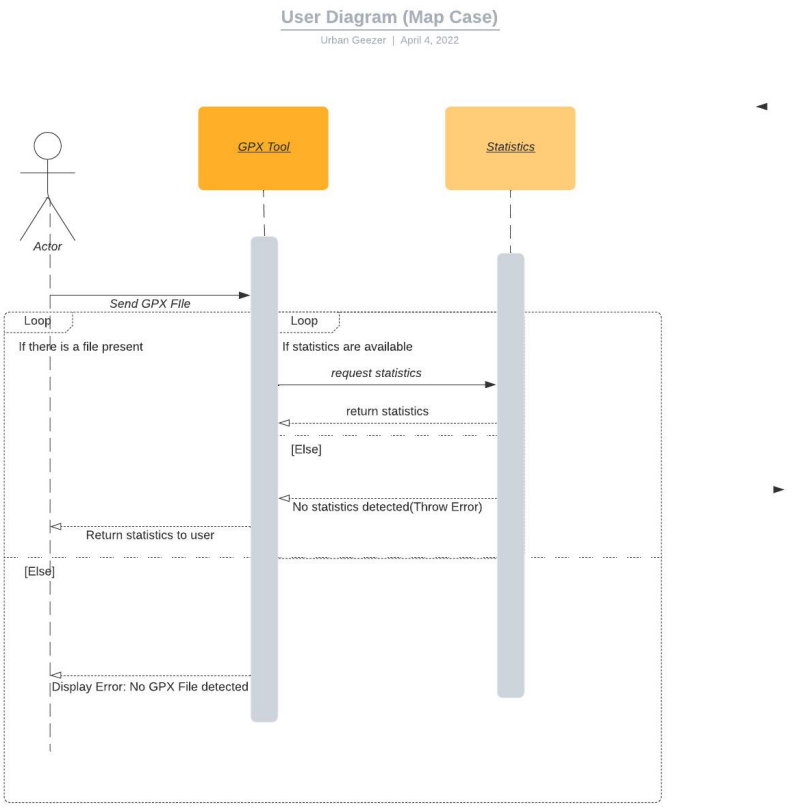
This state machine diagram begins by checking if a file is valid or not. It is part of the GPX parser class. If the file is invalid, the system should return to the first step and request a new, valid file. If the file is valid, the system should go to the Parser state. There, the parser should parse the file to obtain the most important information(the latitude, longitude and elevation) It should then output the statistics in the terminal to the user.

Sequence diagrams

Author(s): Nikolay Filipov, Asim Manzaij



- The first step in a sequence diagram is the actor which in our case is the user that uses our GPX tool. We then have two objects which here are the GPX Tool and the map. The user loads a GPX file which the tool will check if it is a valid GPX file. With this the diagram enters into a loop with the condition of whether the file is valid or not. In this case valid means, if there is enough data to represent a map or if the GPX file is not corrupted. If the GPX file is valid then the GPX tool communicates with the map object and requests a visual of the file. If that is successful the map sends the GPX Tool the visual, which can then be accessed by the user. However, if there is no map the map will communicate with the tool that no map is present. The tool will then throw an error and skip out of the loop to display an error message to the user. If any step of this process is unsuccessful then the GPX Tool will display an error message to the user.



- We have a use case here about the statistics of the GPX file. The actor in our case is the user. The first step in the process is the user needs to upload a GPX file. The process then enters an if loop with the condition of if there is a file present. We made this design choice because if the file is not present or is corrupted the GPX Tool will output an error message to the user. If the file is valid, then the process will enter into another if loop with the condition of are statistics present. If the statistics are present, the GPX Tool will request the statistics from the statistics class and they will be returned to the GPX Tool. After that the GPX Tool will return the statistics to the user. These statistics include time, latitude, longitude, and elevation. If there are no statistics, then the second if loop will throw an error.

Implementation

Author(s): Andres, Alejandro

The strategy we followed in order to move from the UML models to the implementation was at first to create simple diagrams with the basic information we wanted to include in our implementation. Afterwards, we started working on the implementation of these main aspects. When working on this, we realized many other features we needed to include and that would be useful to achieve a better result. Finally, we updated the changes into the UML models so that they would match our current implementation.

After obtaining the total distance of the track we calculated more statistics that we would later use. We created a GUI that consists of various parts; the login, the homepage, a visualization of the GPX file in the form of a map and a window showing the stats of the track that we calculated previously. We accomplished the map visualization thanks to the "JXMapViewer" component of SwingX-WS. We managed to get the JXMap Viewer to work by changing the code and adapting it to ours. It shows the trajectory of the track through a red line and the start and end waypoints with blue markers. There is also the possibility that a third party developer can implement a new sport using the GPXSportPlugin interface.

In this chapter you will describe the following aspects of your project:

- the strategy that you followed when moving from the UML models to the implementation code;
- the key solutions that you applied when implementing your system (for example, how you implemented the movement of the cursor in in the Cyberpunk minigame, how you manage Exploding Kittens matches, etc.);
- the location of the main Java class needed for executing your system in your source code;
- the location of the Jar file for directly executing your system;
- the 30-seconds video showing the execution of your system (you are encouraged to put the video on YouTube and just link it here).

IMPORTANT: remember that your implementation must be consistent with your UML models. Also, your implementation must run without the need of any other external software or tool. Failing to meet this requirement means 0 points for the implementation part of your project.

Main java class: src/main/java/softwaredesign/GUI/LogIn.java
GithubUrl:
https://github.com/latmagan228/Software-Design/blob/3d387db3e3a1c216ecac12787b113a276bbf4659/src/main/java/softwaredesign/GUI/LogIn.java

Jar File: out/artifacts/software_design_vu_2020_jar/software-design-vu-2020.jar
GithubUrl:
https://github.com/latmagan228/Software-Design/blob/fcf32ecfa39f882972791307051e63d4a22d4fa5/out/artifacts/software_design_vu_2020_jar/software-design-vu-2020.jar

YouTube Video Url:
https://youtu.be/DIUOr7Tizo4

Time logs

<Copy-paste here a screenshot of your time logs - a template for the table is available on Canvas>

Team number	41		
Member	**Activity**	**Week number**	**Hours**
Nikolay Filipov	Fixing Diagrams	1	2
Asim Manzaij	Fixing Diagrams	1	2
Alex Pascual	Implementing Map	1	2
Andres Latorre	Implementing Map	1	2
Nikolay Filipov	Diagrams/document	2	2
Asim Manzaij	Diagrams/document	2	2
Alex Pascual	Implementation	2	5
Andres Latorre	Implementation	2	5
Nikolay Filipov	Working on document	3	2
Asim Manzaij	Working on document	3	2
Alex Pascual	Working on document	3	2
Andres Latorre	Working on document	3	2
		TOTAL	30