Assignment 2

Team number: 26
Team members

Name	Student Nr.	Email
Harsh Vijay	2696378	h.vijay@student.vu.nl
Sarthak Bajaj	2699648	s.bajaj@student.vu.nl
Vilen Geghamyan	2703673	v.geghamyan@student.vu.nl
Philip Vacca	2712798	p.p.p.vacca@student.vu.nl

Implemented feature

ID	Short name	Description
F1	User Profile	The user can enter their name, age, gender, height and weight to make it more personal.
F2	Extendibility	A developer can add different sports with ease by specifying some basic characteristics about the specific sports.
F3	Command-line interface	The implementation uses a simple command-line interface to ask users for input and to display results. It is interactive and also easy for any user to use.
F6	Parsing the GPX file	Users can see various analyses which can be selected after selecting the sports activity and uploading the gpx file. ● Average Speed. ● Total Covered Distance ● Time Duration of the activity ● Their location timeline during the exercise
F8	History of Tasks	The history of tasks(the results of the calculations) will be stored in the metrics buffer class.
F9	Exit	The user can exit out of the programme at any time by typing 'Exit' and hitting enter.

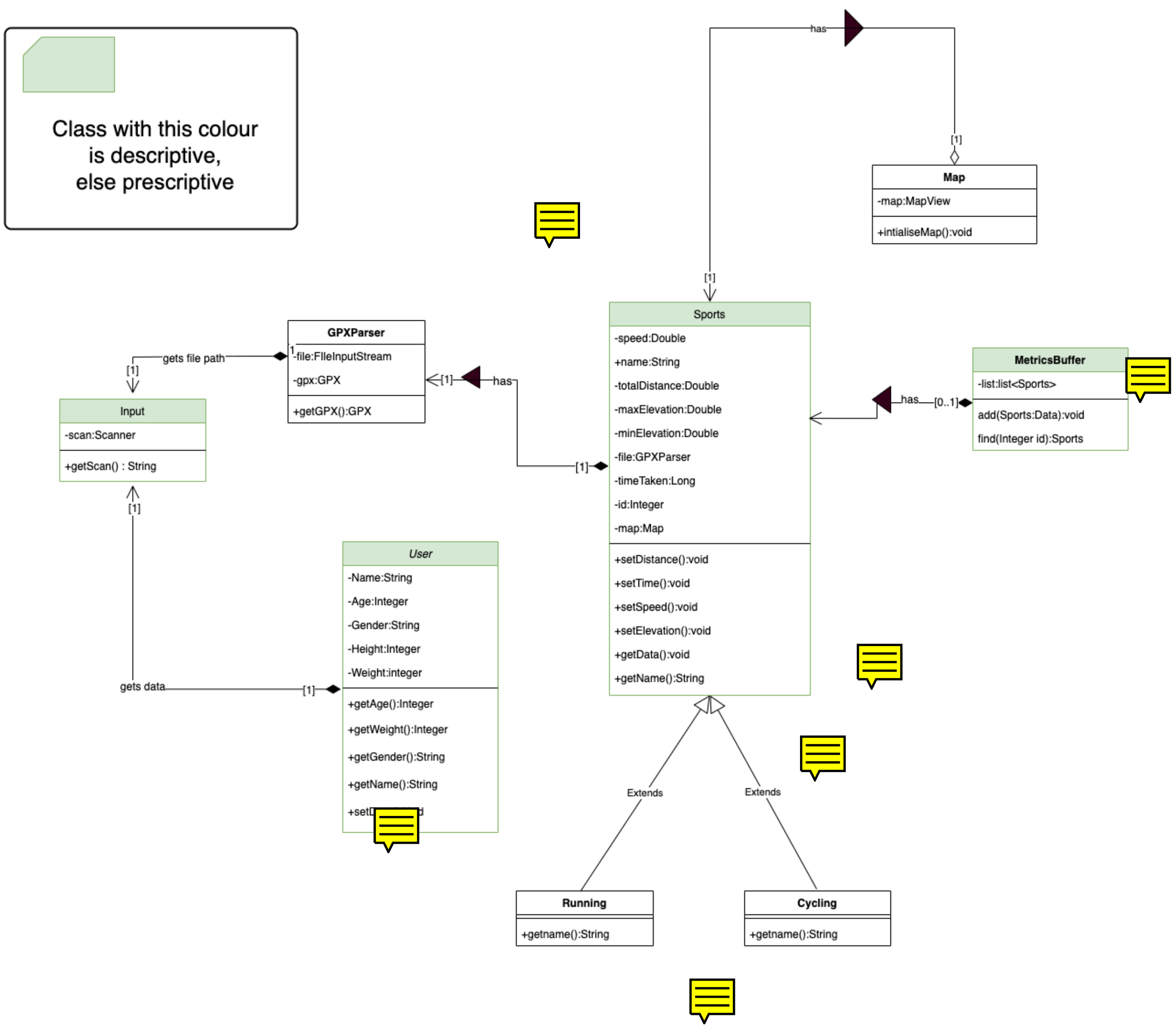
Used modelling tool: diagram.io

Class diagram

Author(s): Sarthak Bajaj

This chapter contains the specification of the UML class diagram of your system, together with a textual description of all its elements.

Figure representing the UML class diagram of GPX Manager



MetricsBuffer

This class is used to store all the history of tasks in a form of a list of datatype Sports which is a superclass for all the sports that are implemented.

Attributes

- ***list-*** A list of sports that contain the metrics of all the gpx files that have been parsed during the execution of the program.

Operations

- ***add -*** This adds an instance of sports into the list.
- ***find -*** this finds the sports data from the list with the unique id which is provided as a parameter while calling the function.

Metrics buffer has a shared aggregation relation with sports because metrics buffer stores instances of sports as a list that means metrics buffer cannot exist without the existence of sports class.

GPXParser

GPXParser is a class that uses an external library to parse the gpx file. Which helps the program to calculate different features and visualise the gpx file into a map.

Attributes

- ***file-*** This attribute creates a file input stream that helps to read the desired file from the system.
- ***gpx-*** This attribute creates an instance of GPX class from the library which parses the gpx file into its subcomponents.

Operations

- ***getGPX -*** Returns the parsed data of gpx file, of datatype GPX(Class from the jpx library)

Sports

Sports is an abstract class that is extended by running and cycling. Sports class represents the metrics of the gpx file for the sport such as speed,time taken,etc. The class is abstract because it provides the basic attributes for every sport and also gives an extensible feature to the code which will allow a developer to extend the number of sports without any issue.

Attributes

- ***speed-*** Represents the average speed of the user during the entire activity.
- ***name-*** Represents the name of the sports which will be changed by its subtypes according to their names. In the abstract class, this attribute is initialised as “sports”.
- ***totalDistance -*** Represents the total distance covered during the activity.

- ***timeTaken -*** The total time taken to finish the activity.
- ***maxElevation -*** The maximum altitude during the course of the activity.
- ***minElevation-*** The minimum altitude during the course of the activity.
- ***file-*** Parsed gpx file that will be used to calculate attributes such as speed, distance, time, etc.
- ***id -*** a unique id that will help find the data from the list of history of tasks.
- ***map -*** Instance of map class which will give the visual representation of the gpx file .

Operations

- ***setDistance -*** calculates the total distance traversed during the activity.
- ***setTime-*** calculates the total time taken to complete the activity.
- ***setSpeed -*** calculates the average speed during the entire activity.
- ***setElevation -*** calculates the minimum and maximum elevation during the entire activity.
- ***getData -*** prints the data such as speed, distance, time taken, minimum and maximum elevation.
- ***getName -*** returns the name of the sport. For example, cricket, running, etc.

Sports has a composition relation with GPXParser because sports class cannot exist without gpxparser parsing the file for sports to calculate the attributes.

Map

As the name suggests this class deals with the creation of maps and their objects such as tracks and points. It uses an external library to help build maps.

Attributes

- ***map-*** Creates an instance of the Mapview class which creates a map to visualise the gpx file. For example, a route around Vondelpark would be depicted on the map.

Operations

- ***initialiseMap -*** This function will initiate the process of visualisation of a map with the points provided in the gpx file.

Map has a composition relation with Sports and because map cannot exist without data provided by sports.

Input

This class deals with all the input that we take from the user. It takes the input from standard input using the Scanner Function of java. That scans the System.in every time it is called.

__Attributes__

- ***scan -*** An instance of scanner that deals with all the input coming from Standard input can also be termed as the console of the java IDE.

__Operations__

- ***getscan -*** it returns the input given by the user into the standard input stream in the form of String. Which will be used by other classes to gather data or to calculate different attributes.

Input has a composition relation with both GPXParser and User class because they need input to initialise both classes which means they cannot exist without Input class

__Running__

Running is a subtype of Sports that has all the attributes of sports and can be initialised by the user. The initialization of this class depends on the type of sports chosen by the user. It is initialised whenever the user types running when given an option to choose a sport.

__Operations__

- ***getName -*** returns running as a string.

__Cycling__

Cycling is a subtype of Sport that has all the attributes of sports and can be initialised by the user. The initialization of this class depends on the type of sports chosen by the user. It is initialised whenever the user types cycling when given an option to choose a sport.

__Operations__

- ***getName -*** returns cycling as a string.

__User__

The **User** class takes an input from an user to make it more personalised.

__Attributes__

- **Name** - It takes in the user name that they wish to be called by the GPX manager.
- **Age** - It takes the age of the user as an Integer.
- **Gender** - It takes in the gender of a person as a string.
- **Height** - It takes in the height of a person as an integer in centimeters.
- **Weight** - It takes in the weight of a person as an integer in kilograms.

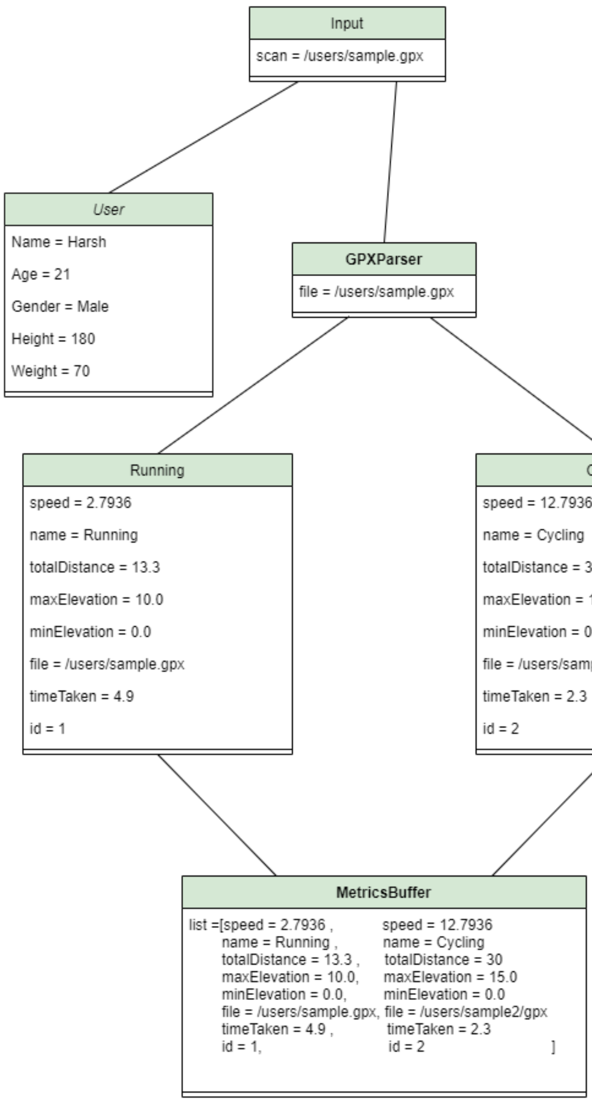
Operations

- ***getAge()*** - It returns the age of the user they put into the manager
- ***getWeight()*** - It returns the age of the user they put into the manager
- **getGender()** - It returns the gender of the user they put into the manager
- ***getName()*** - It returns the name of the user they put into the manager
- ***setData()*** - It is a method a function that asks the user to input their data such as name, age etc.

Object diagram

Author(s): Harsh Vijay

This chapter contains the description of a "snapshot" of the status of your system during its execution. This chapter is composed of a UML object diagram of your system, together with a textual description of its key elements.

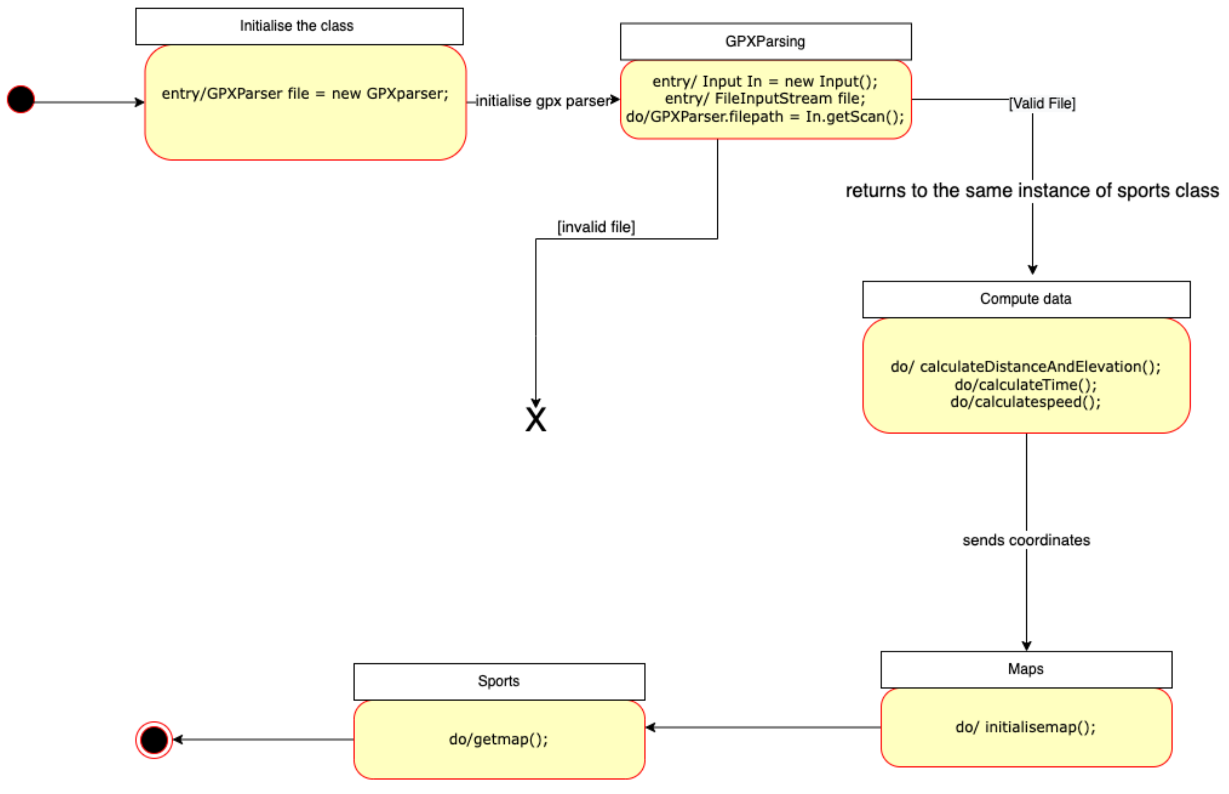


The diagram shows the object diagram for the GPX manager. The purpose of this diagram is to better illustrate, with an example, how the classes described in the Class Diagram will interact with each other. We start with an instance of an *Input* which is taking the input of a file path "/users/sample.gpx".Then the Input is extended to the *User* class where its instances are taking the name, age, gender, height and weight of the user. The *Input* class also extends to *GPXParser* where it is getting the file path that has been put in the *Input* instance which will creates a file input stream and will parse the gpx file with the use of an external library. The *GPXParser* extends to two different classes *Running* and *Cycling* where instances represent the metrics of the gpxfile taken from the file "/users/sample.gpx" which has been parsed from the *GPXParser*. *Running* and *Cycling* instances are showing all the metrics data which is then stored into the *MetricsBuffer* class in a list. *MetricsBuffer* class is storing all the instances of Running and Cycling which will be used to find or add more data.

State machine diagrams

Author(s): Vilen Geghamyan

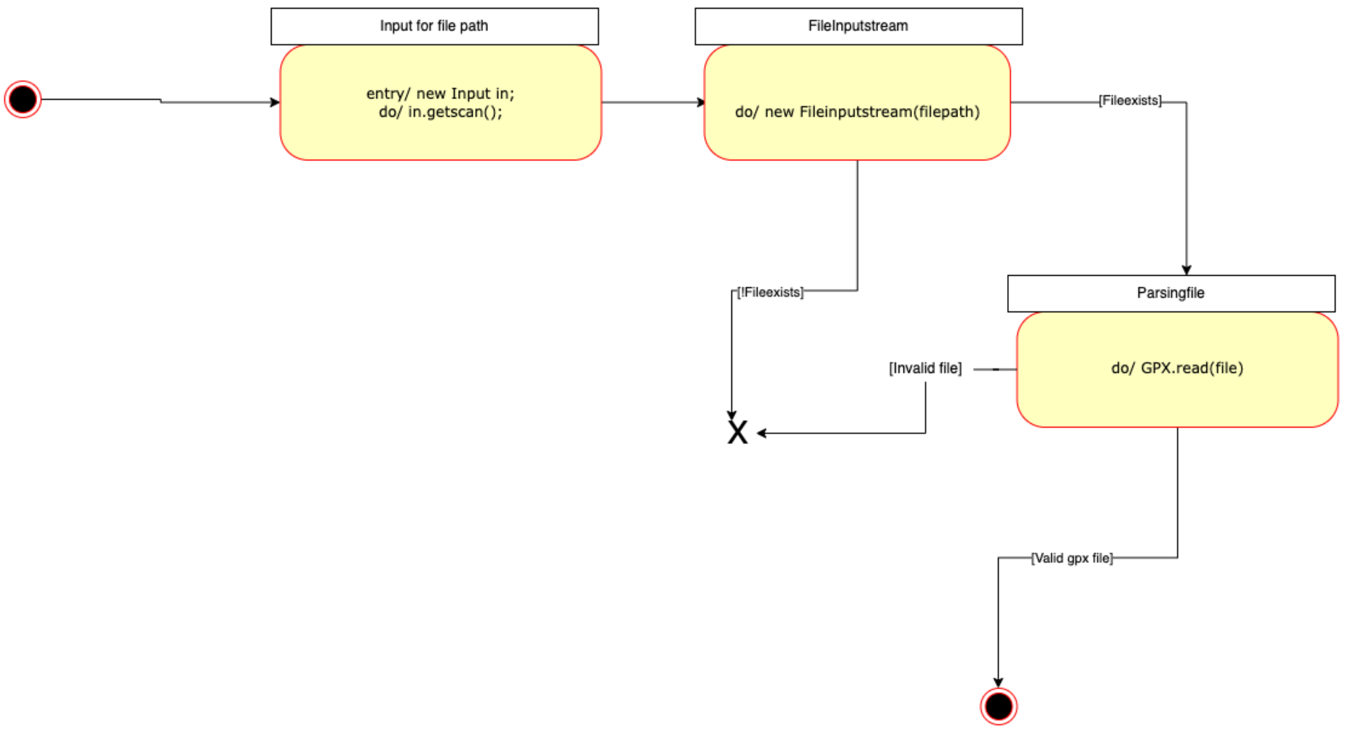
1.


Initialise the class
entry/GPXParser file = new GPXparser;
initialise gpx parser
GPXParsing
entry/ Input In = new Input();
entry/ FileInputStream file;
do/GPXParser.filepath = In.getScan();
[Valid File]
returns to the same instance of sports class
[invalid file]
X
Compute data
do/ calculateDistanceAndElevation();
do/calculateTime();
do/calculatespeed();
sends coordinates
Maps
do/ initialisemap();
Sports
do/getmap();

This state machine diagram serves as an overview of all the possible states that {abstract} “Class Sports” can be in.

In the first state we Initialise the class by creating a new object instance of another class *GPXParser*. This transits us to the next state which is **“**GPXParsing**”** which will serve as ***GPXParser*** initialization. When entering the state we create two instance objects, one for class ***Input*** and another for class ***FileInputStream***. We use these two objects to take the gpx file path as an input stream and parse the file. If the file format is different from “gpx” it is considered as an “invalid file” and the operation is terminated. Else if the file is of “gpx” format, it is validated and transits back to the same instance of sports class in the new state “Compute Data”. While in this state, we do several metrics data calculations. From those calculations we will get the coordinates parsed from the gpx file and send them as data to the class ***Maps***. While in that state, the data containing the coordinates will be as parameters to the *initialisemap()* function, which will initialise our map. After the initialisation we change the state back to class “Sports” where we use the function *getmap()* to visualise the map obtained from the gpx file.

2.

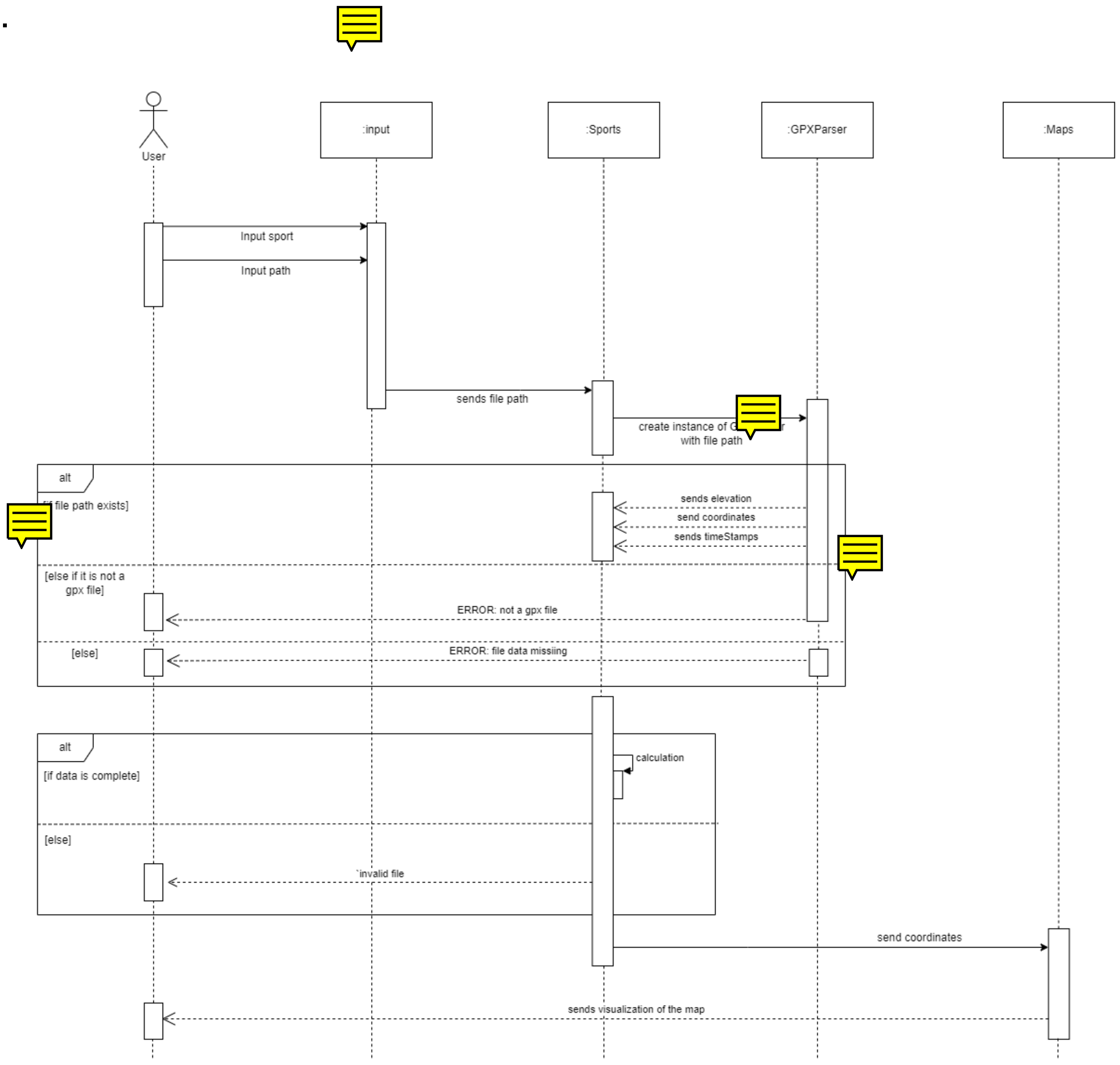


In the second state machine diagram, we describe the *GPXParser* class. We begin with creating a new object of class “Input” which we use to initialise an input for the file path. In the next state we create a new instance of “FileInputstream” with argument the filepath and then we check if the file exists we transit to the new state “Parsingfile”, if the file does not exist, we terminate. While being in the new state “ParsingFile” we check the type of the parsed file. If the file is of type gpx, the file is validated and we reach to our end state, else, if the file is not of type gpx, we get an invalid file error and if the file is valid it parsed and broken down into sub components which is further use to carry out calculation and visualize into map by sport class.

Sequence diagrams

Author(s): Harsh Vijay and Philip Vacca

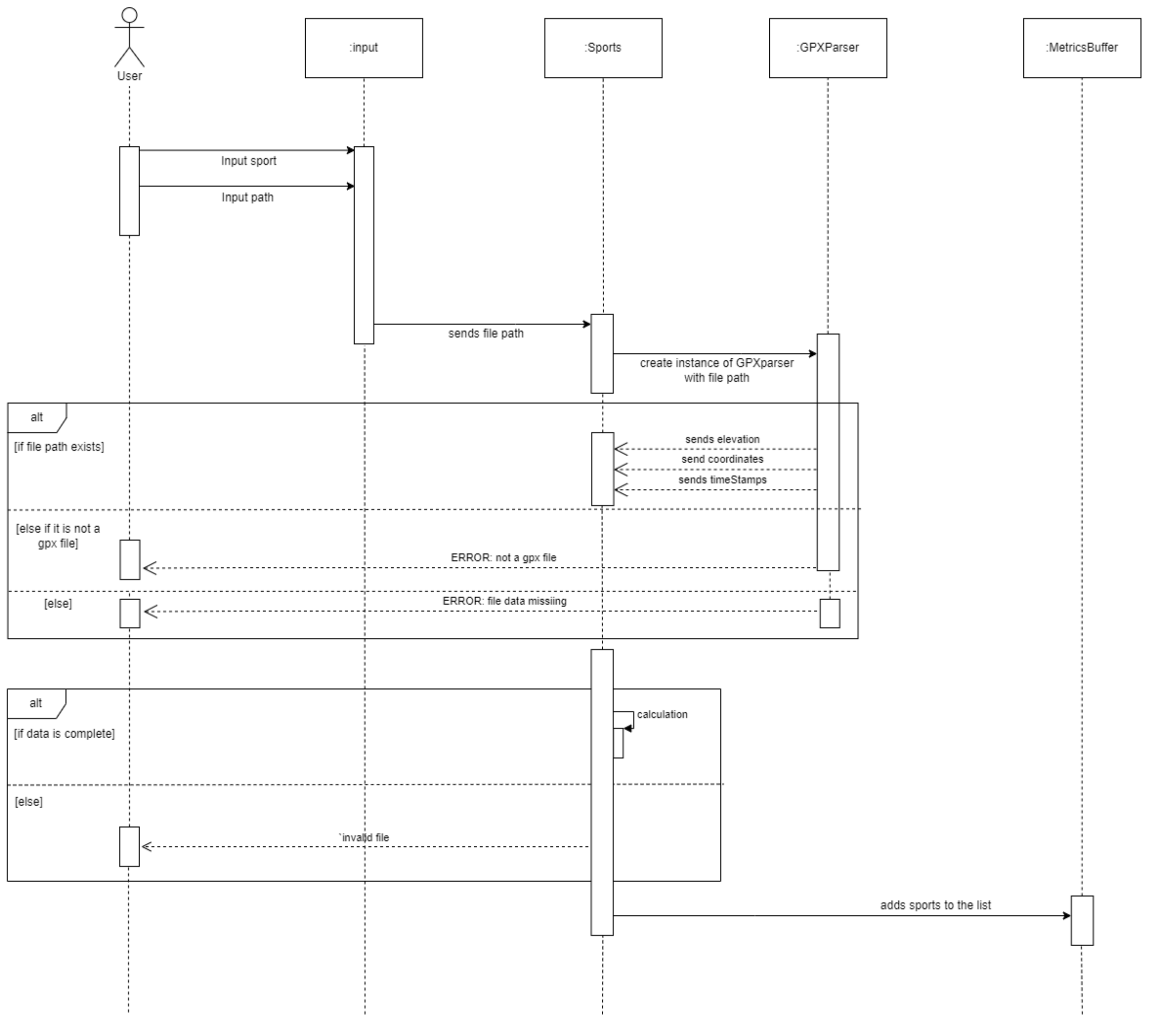
1.



This sequence diagram describes how the user input is processed, parsed and turnt into a map that can be seen by the user. The first event described in the sequence diagram above is the user inserting the file path of the GPX file that contains the sporting data, together with the name of the sport.

The file path and the name of the sport will be inserted into the *input* class, whereafter it will send the file path to the Sports class. Within the sports class, an instance of the object GPXparser with the file path is created. Inside the *GPXParse*r object, the file path will be checked: If the file path does not exist then it will send a return message to the user telling them that the file path they entered does not exist. If the file path exists but it does not point to a GPX file, then a return message will be sent to the user telling them that the file that corresponds to the file path they entered does not point to a GPX file. In both cases the user is prompted to submit a valid file path. If the file path exists and it is indeed pointing to a GPX file, the file that belongs to the corresponding file path will be parsed and the elevation, coordinates (longitude and latitude) and timestamps will be sent back to the *Sports class*. Here the data will be checked on its completeness; If the data is incomplete*(e.g., timestamps missing)* then a return message will be sent to the user asking them to submit the file path of a complete GPX file. If the data is complete, the longitude and latitude will be sent over to the Maps object where the longitude and latitude are loaded onto a map. Thereafter the visualisation of the map will be sent back to the user.

2.



The first part of this sequence diagram is the same as the previous one, where it is shown how the input data is processed and where it is parsed. This sequence diagram describes how the input data is turned into metrics that will be stored in a buffer. After the data is parsed and has been sent back to the *Sports* class, the data will be checked on completeness: If the data is incomplete*(e.g., coordinates missing)* then an error message about incomplete data and the programs terminates. If the data is complete, it will be used to calculate the metrics. First, the total distance is calculated by finding the distance between consecutive individual points and then adding them together. Secondly, the total time is calculated by subtracting the time of the first trackpoint to the time of the final point. Then, the elevation is calculated with the use of a simple algorithm: both the min and max elevation variables are first set to the elevation of the first trackpoint, then they will continuously be updated while traversing through the elevations of each track point by checking whether the elevation of that point is less than or greater than the elevation variables respectively. When all trackpoints have been traversed, those variables contain the true min and max elevation values for the activity. After these metrices have been calculated, they are sent to the MetricsBuffer class where they will be stored.

Implementation

Author(s): Sarthak Bajaj

Our focus, when we started designing the class diagrams, was to make the system extensible and also easy to understand. While progressing through the diagrams there were many changes that were done due to the variety of challenges that came up during the implementation of the design. We started with just trying to parse a gpx file. Which was not easy in the first place, Much of our time was consumed while trying to figure out how can we access the data has been parsed. Reading through the readme file of the library that we use to parse gpx gave us an idea of how we can access different attributes of the parsed data.

Our strategy while implementing the code was to design a basic version that satisfies the requirements and then implement it. If we faced any problem, we refer back to design and see what we can change to improve our design in a way that terminates the problem we are facing while keeping in mind the extensibility of the code.

Our solutions are as follows:

- Storing the history of tasks in a list. In our implementation, we used a list to store all the instances of the subtypes of Sports. We have two basic functions in the class first to add() something to the list and second to find() that helps to find the instance with a unique id. In our implementation, we provide every instance of sports with a unique id which makes it easy to find from a list of data.

- We ask the user to provide the path of the file, which is stored as a string and then used to initialize a file input stream that checks if the file exists or not. We initialize the gpxparser with the file input stream which returns the parsed version of the file. The parsed file is then used to calculate different attributes of sports and also visualize the maps.

The location of the main java class is
src/main/java/softwaredesign/main.java

The location of the jar file
out/artifacts/software_design_vu_2020_jar/software-design-vu-2020.jar

Link to the video

Time logs

Team number	26		
Member	**Activity**	**Week number**	**Hours**
Group	correction for Assignment 1	3	2
Group	Further outlining system	3	3
Sarthak and Vilen	class diagram	4	2
Harsh and Philip	understanding the class diagram	4	1
Vilen	state machine diagram	4	3
Sarthak	class diagram	4	1
Harsh and Philip	sequence diagram	4	2
Harsh	object diagram	5	2
Philip	sequence diagram	5	2
Group	Implementation	5	5
Sarthak	Final Implementation	5	2
Group	Meeting for the report	5	1
Group	writinng the report	5	3
Group	Formatting the report	5	1
		TOTAL	30