Shape

Description automatically generated

Systemdokumentation

zu

**myTracks**

Version 1.0 vom 31.07.2022

Entwickelt von Huber Andreas, Mathias Schwarz und Markus Schwarz

Unter Aufsicht von a. Univ.-Prof. Dr. Johannes Sametinger

**Inhaltsverzeichnis**

[1 Verwendete Werkzeuge, Bibliotheken und Technologien 3](#_Toc110086219)

[1.1 Werkzeuge 3](#_Toc110086220)

[1.2 Bibliotheken 3](#_Toc110086221)

[1.3 Technologien 4](#_Toc110086222)

[2 Systemvoraussetzungen 5](#_Toc110086223)

[3 Paradigmen 5](#_Toc110086224)

[3.1 Benutzerfreundlichkeit der Applikation: 5](#_Toc110086225)

[3.2 Objektorientierung des Codes: 5](#_Toc110086226)

[3.3 Wiederverwendbarkeit des Codes: 5](#_Toc110086227)

[4 Klassenhierachie 6](#_Toc110086228)

[4.1 GPSTracker: 6](#_Toc110086229)

[4.2 Loader: 6](#_Toc110086230)

[4.3 GPSTrackerGUI: 7](#_Toc110086231)

[4.4 Logging: 7](#_Toc110086232)

[5 Abbildungsverzeichnis: 8](#_Toc110086233)

[6 Quellen 8](#_Toc110086234)

1. Verwendete Werkzeuge, Bibliotheken und Technologien
   1. Werkzeuge

Grundlegen wurde das gesamte Projekt in der IDE IntelliJ IDEA 2021.3 (Ultimate Edition) in der Educational- Version entwickelt. Für kooperatives Zusammenarbeiten der Projektmitglieder wurde ein GitHub- Projekt erstellt und in Kombination mit GitHub Desktop und dem zugehörigen IntelliJ- Plugin verwendet. Für die Zeitaufzeichnung wurde ein Clockify- Projekt erstellt mit dessen Hilfe die Teammitglieder ihre Projektzeiten und Tätigkeiten getrackt haben.

Die Visualisierung und Kommunikation erfolgte anfangs über MiroBoard und Jira, verlagerte sich in den späteren Projektphasen allerdings mehr in Richtung Discord.

Zur Analyse des Codes wurde Sonarlint in der aktuellsten Version verwendet, da die Rezensionen und Erfahrungsberichte weitaus positiver ausfielen als bei dem vergleichbaren Plugin SonarCube.

(IntelliJ; Github; Clockify; MiroBoard; Jira; Discord; Sonarlint)

* 1. Bibliotheken

**JFreeChart 1.0.19**

Für die Visualisierung der Barcharts wurde die Open Source- API JFreeChart in Kombination mit JCommon verwendet. Das Chart wird entsprechend der Benutzereingabe mit unterschiedlichen Datensätzen beschrieben und angepasst.

**Sax Parser**

Wird zum Einlesen der Informationen aus den GPS- Datrensatz- Files verwendet. Unser ActivityHandler erweitert den DefaultHandler und überschreibt die abstrakten Default- Methoden, um die Datensätze aus den GPS- Files einzulesen. Der Sax Parser ersetzte im Verlauf der Entwicklung, den zuerst verwendeten Dom Parser da dieser effizienter arbeitet und so die Wartezeit des Benutzers empfindlich verringert.

**Dom Parser**

Wird zum Schreiben bzw. Transformieren des Inhalts des Properites.xml- Files verwendet. In diesem File wird aktuell nur der Pfad, aus dem die GPS- Files eingelesen werden, persistiert.

**Standard Java Bibliotheken**

Alle restlichen Methoden und Funktionen wurden mit den standardmäßig in Java enthaltenen Bibliotheken umgesetzt.

(JFreeChart; Sax Parser; Dom Parser)

* 1. Technologien

Die gesamte Applikation wurde auf Basis von Java Version 17 und JUnit 5.11.1 entwickelt. Für die grafische Darstellung wurde Java Swing gewählt. Im Hauptfenster (JFrame) wurden dazu direkt in IntelliJ ein JMenu und mehrere JPanels mit den jeweiligen weiter untergliederten Inhalten erstellt.

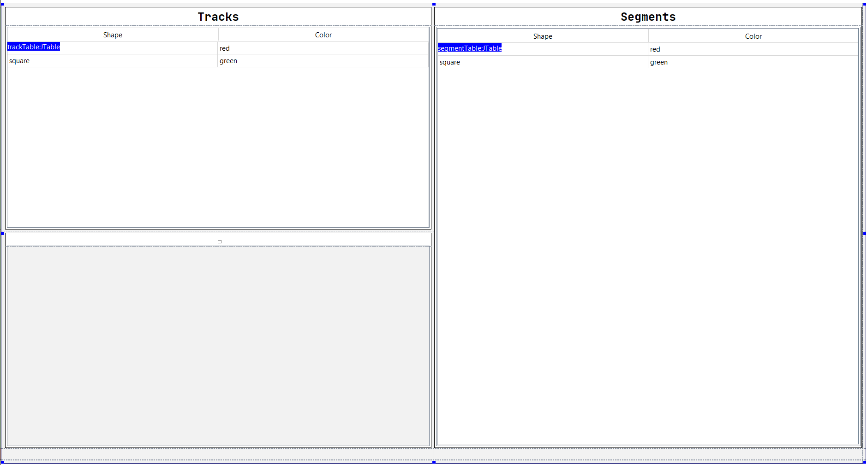


Abbildung 1: Aufbau der grafischen Oberfläche

**1**

*rootPanel:* übergeordnetes JPanel das 3 weitere JPanels beinhaltet

**4**

barChartPanel: enthält ein JLabel und ein weiters JPanel in dem das Barchart eingebettet ist

**3**

segmentPanel: enthält ein JLabel und ein JScrollPane in dem sich eine JTable mit Segmenten befindet

trackPanel: enthält ein JLabel und ein JScrollPane in dem sich eine JTable mit Tracks befindet

**2**

1. Systemvoraussetzungen

Voraussetzungen für das Ausführen der Applikation sind lediglich eine installierte Version von Java Runtime, etwa 3,5 MB Speicherplatz für die Jar-Datei und eine, für die heutige Zeit, nicht nennenswerte CPU- Leistung. Im laufendenden Betrieb werden durchschnittlich etwa 300 MB Arbeitsspeicher von der Applikation belegt. Abhängig von der Anzahl und Länge der zu lesenden GPS- Daten müssen auch hier nur minimale Speicherressourcen belegt werden. Bei den verwendeten Testdaten kann von etwa 0,33 MB pro File ausgegangen werden.

1. Paradigmen

Um ein möglichst konsistentes und qualitativ hochwertiges Ergebnis zu erhalten, haben wir uns hauptsächlich auf drei Grundziele geeinigt.

* 1. Benutzerfreundlichkeit der Applikation:

Primäres Ziel des Projekts war und ist für uns die Benutzbarkeit der entwickelten Applikation. Es wurde daher besonderes Augenmerk auf das grafische Design und die Lesbarkeit der Informationen gelegt. Deshalb wurde beispielsweiße auch der Loading- Screen eingeführt, um dem Nutzer zu signalisieren, dass der Programmstart erfolgt ist und die Zeit, welche zum Einlesen der Daten benötigt wird, überbrückt wird.

* 1. Objektorientierung des Codes:

Die Objektorientierung des Codes wurde im Sinne von der Organisation in Klassen, welche miteinander kommunizieren umgesetzt. Der lesende und schreibende Zugriff auf Klassenvariablen soll nur über die definierten Schnittstellen erfolgen.

* 1. Wiederverwendbarkeit des Codes:

Weiters war es und von Anfang an ein Anliegen implementierte Methoden so zu gestalten, dass diese wiederverwendet, werden können und Code- Duplikate weitestgehend vermieden werden. Methoden wurden also so designend, dass diese beim Programmstart ebenso verwendbar sind wie bei Aktualisierungen im laufenden Betrieb.

1. Klassenhierachie

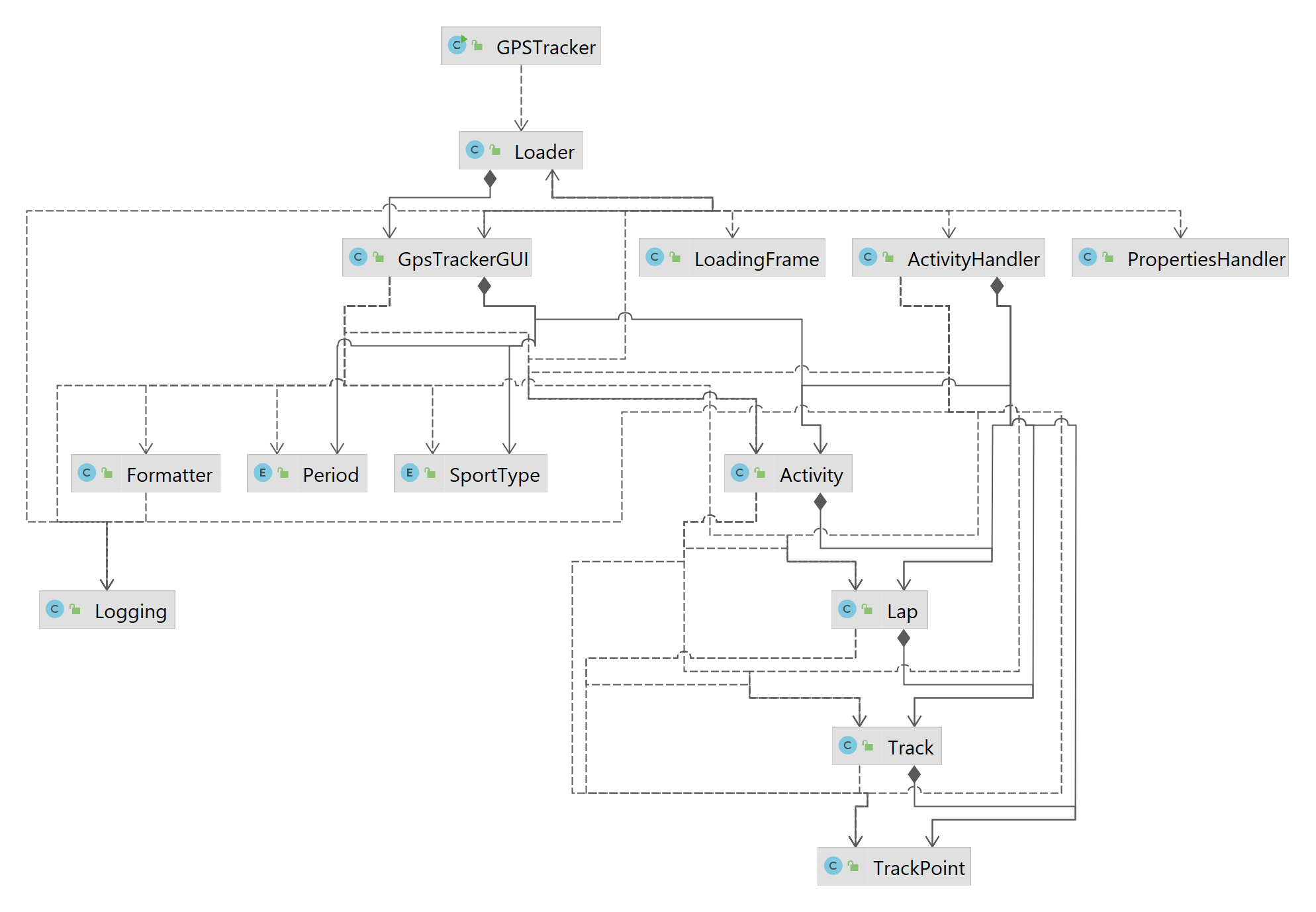


Abbildung 2: Klassenhierachie

Im Folgenden werden die wichtigsten Klassen beschrieben, um einen Überblick über die Abhängigkeiten und Funktionsweißen der einzelnen Bestandteile zu geben. Zusätzlich kann das mitgelieferte JavaDoc herangezogen werden, um die Funktionen und Parameter der einzelnen Methoden und Variablen nachzuvollziehen.

* 1. GPSTracker:

In der Main- Klasse *GPS- Tracker* wird nur die Klasse *Loader* aufgerufen.

* 1. Loader:

In der *Loader*- Klasse wird die Klasse *LoadingFrame* aufgerufen welche einen Lade- Screen zeigt, um dem Nutzer zu signalisieren, dass die Applikation gestartet wurde, während die zu verwertenden Daten eingelesen werden.

Weiters wird die Klasse *PropertiesHandler* aufgerufen welche auf ein lokal gespeichertes XML- File zugreift, beziehungsweiße erstellt welches den Speicherort der GPS-Datensätze enthält.

Anhand der gelesenen Properties wird die Klasse *ActivityHandler* aufgerufen welche die GPS- Datensätze einlest und diese in interne Listen speichert. Diese sind in *Acitivities*, *Laps*, *Tracks* und *Trackpoints* gegliedert.

* 1. GPSTrackerGUI:

In *GPSTrackerGUI* wird die grafische Oberfläche mit den Tabellen, dem Barchart und Menüpunkten erzeugt. Sie greift lesend auf die Klassen *Acitivities*, *Laps*, *Tracks* und *Trackpoints* zu und füllt so die Tabellen. Bei den Klassen *Period* und *SportType* handelt es sich um enum- Klassen welche die Standardwerte für die Filteroption/ Gruppierung nach Sportart und Zeitraum enthalten.

Für die Darstellung der Daten innerhalb der Tabellen wurde die *Formatter*- Klasse angelegt, welche die unterschiedlichen Informationen sinnvoll formatiert, um ein gut lesbares Ergebnis zu erzielen.

* 1. Logging:

Die *Logging*- Klasse dient nur zur Unterstützung der Entwicklung, und erlaubt es Konsolenausgaben zum Applikationsstatus, über ein Flag, zu aktivieren bzw. zu deaktivieren.

1. Abbildungsverzeichnis:

[Abbildung 1: Aufbau der grafischen Oberfläche 4](#_Toc110086100)

[Abbildung 2: Klassenhierachie 6](#_Toc110086101)

1. Quellen

Clockify. Online verfügbar unter https://clockify.me.

Discord. Online verfügbar unter https://discord.com.

Dom Parser. Online verfügbar unter https://docs.oracle.com/javase/tutorial/jaxp/dom/readingXML.html.

Github. Online verfügbar unter https://github.com.

IntelliJ. Online verfügbar unter https://www.jetbrains.com/de-de/idea/.

JFreeChart. Online verfügbar unter https://www.jfree.org/jfreechart/.

Jira. Online verfügbar unter https://www.atlassian.com/de/software/jira.

MiroBoard. Online verfügbar unter https://miro.com/de/.

Sax Parser. Online verfügbar unter https://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html.

Sonarlint. Online verfügbar unter https://www.sonarlint.org.