a) **Zielsetzung** : Die SpeedCamera überwacht die Einhaltung der Geschwindigkeitsbegrenzung von 50 km/h innerorts.

b) **Aufbau / Struktur** : Die **SpeedCamera** besteht aus drei aufeinandergestapelten Section. Die untere Sektion beheimatet die **CentralUnit**. Die mittlere Sektion beheimatet einen **LaserScanner**, die obere Sektion beheimatet eine **LED** in der Farbe RED.

Der CentralUnit ist ein **Storage registeredOfficer** – realisiert als TreeMap – für die registrierten Officer sowie ein **Storage fineRecords** – realisiert als ArrayList – für die erfassten Geschwindigkeitsverstöße zugeordnet.

Auf dem Deckel der oberen Sektion ist das MobileNetworkModule installiert.

c) **Charakteristika** : Die SpeedCamera ist charakterisiert durch serialNumber vom Typ UUID, manufacfuringDate (aktuelles Datum im Format dd.MM.yyyy).

Ein (abstrakter) **Human** ist charakterisiert durch name, birthDate und face. Ein **Owner** ist ein Human und zusätzlich charakterisiert durch SmartPhone und einem zugeordneten Car. **SmartPhone** ist charakterisiert durch phoneNumber und Wallet. Zwischen SmartPhone und Wallet existiert eine physische Inklusion. Ein **Wallet** ist charakterisiert durch deposit mit einem initialen Wert von 5000. Einem Owner ist genau ein SmartPhone mit einer eindeutigen mobileNumber zugeordnet. Ein **Officer** ist ein Human und zusätzlich charakterisiert durch id und eine **IDCard**. Die IDCard hat einen magneticStrip zwecks Speicherung der mit SHA256 verschlüsselten vierstelligen nummerischen pin. Zwischen IDCard und magneticStrip existiert eine physische Inklusion.

Bei der **MobileCentralUnit** – realisiert als Enumeration – sind alle SmartPhone der Owner in einer hash-basierten Datenstruktur registriert.

Ein **Car** ist charakterisiert durch manufacturer, registrationID und speed.
Einem Car ist ein LicensePlate zugeordnet. LicensePlate ist charakterisiert durch id.

Die **VehicleRegistrationAuthority** verwaltet die Zuordnung Owner und Car in einer hash-basierten Datenstruktur zwecks Abfrage auf Basis des licensePlate.

Die **Police** ist charakterisiert durch wantedOwner, arrestedOwner, confiscatedCars.

Die Informationen bezüglich Car, LicensePlate und Owner sind der Datendatei „data.csv“ zu entnehmen.

d) Auf einem **ParkingSpace** (50x20) werden 1000 Car platziert.

e) Der Bußgeldkatalog (https://www.bussgeldkatalog.de/geschwindigkeit/) nur mit den Bußgeldern in Euro ist in einer Konfigurationsdatei im Format JSON definiert. Punkte und Fahrverbot werden aus Aspekten der Vereinfachung nicht importiert/berücksichtigt.
Im Rahmen des Import wird diese JSON-Datei eingelesen.

f) **Steuerung** : Die Applikation wird zentral über eine Menüstruktur **[01]** Startup, **[02]** Import, **[03]** Execute Simulation, **[04]** Report, **[05]** Export, **[06]** Shutdown und **[07]** Exit.

g) Bei Aufruf von **Startup** wird der ParkingSpace mit den Car initialisiert und die SpeedCamera gestartet (isShutdown = false). Die Owner „sitzen“ in den Car und warten auf Abruf.

h) Der Aufruf von **Import** ist möglich, wenn die SpeedCamera gestartet wurde. Es werden über die Konsole die id sowie das 4-stellige nummerische password des Officer abgefragt. Die Kombination id und password wird von der CentralUnit auf Korrektheit geprüft. Bei nicht erfolgreicher Authentifizierung erfolgt die Fehlermeldung „credentials incorrect“ und Rücksprung zum Menü. Bei erfolgreicher Authentifizierung erfolgt die Rückfrage, ob der Import durchgeführt werden soll. Bei Bestätigung mit y erfolgt der Import der Datendatei fine_catalogue.json aus einem definierten Datenverzeichnis. Bei Abbruch mit n erfolgt Rücksprung zum Menü. Nach Import steht das Regelwerk der FineEngine zur Anwendung zur Verfügung.

i) Bei Aufruf von **Execute Simulation** wird die Simulation ausgeführt. Die Simulation besteht aus 100 Iterationen. Für jede Iteration werden 100 Fahrzeuge von dem ParkingSpace zufällig ausgewählt und in eine Warteschlange – realisiert als LinkedList – eingereiht.

   Nach Initialisierung passieren die Fahrzeuge mit einer zufällig bestimmten Geschwindigkeit im Bereich 45-120 sukzessive die SpeedCamera.

   Die Wahrscheinlichkeit einer Geschwindigkeitsüberschreitung (> 53) liegt bei 10%.

   Im Falle einer Geschwindigkeitsüberschreitung wird folgende (vereinfachte) Logik angewandt. 21-70 km/h (zufällig bestimmt) mit einer Wahrscheinlichkeit von 15%, sonst 10-20 km/h (zufällig bestimmt).

   Der LaserScanner erfasst zu jedem Car die Geschwindigkeit und leitet diese Information an die FineEngine zwecks Prüfung weiter. Im Falle einer Geschwindigkeitsüberschreitung werden von der FineEngine die rote LED und die Camera ausgelöst. Die Camera erfasst die FrontSide des Car als Picture[1] gemäß dem spezifizierten Format sowie den Zeitstempel in Nanosekunden und übermittelt diese Informationen an die FineEngine.

   Die FineEngine beauftragt die AIEngine das face sowie das licensePlate aus dem Picture zu extrahieren. Mit dem extrahierten face versendet die FineEngine über das MobileNetwork-Module der SpeedCamera eine mit AES verschlüsselte Anfrage an die Police, ob der Owner zur Fahndung ausgeschrieben ist und ein Haftbefehl vorliegt. Mit dem extrahierten licensePlate versendet die FineEngine über das MobileNetworkModule der SpeedCamera eine mit AES verschlüsselte Anfrage an die VehicleRegistrationAuthority zwecks Ermittlung der Daten zum Owner (name, birthDate und phoneNumber). Das Ergebnis der Abfrage wird ebenfalls mit AES verschlüsselt und über das MobileNetwork-Module empfangen.

   Die FineEngine erstellt ein Record (fortlaufende sequenceID beginnend bei 1, timestamp in Nanosekunden, timestamp im Format dd.MM.yyyy hh:mm:ss, picture, licensePlate, name, birthDate, phoneNumber, allowedSpeed, measuredSpeed, measuredSpeedAfterDeducting-Tolerance, penalty). Der Wert zu penalty bestimmt sich aus der gemessenen Geschwindigkeit abzüglich 3% Toleranz sowie der Tabelle aus der importierten Datendatei fine_catalogue.json. Die FineEngine hat über das MobileNetworkModule eine Verbindung zur MobileCentralUnit. Über die phoneNumber des Owner verbindet sich die FineEngine mit dem SmartPhone des Owner und greift auf das Wallet zu. Das Wallet wird mit der penalty belastet.

1 Beispiel : example_picture.txt

Bei einem zur Fahndung ausgeschriebenen Owner wird das Fahrzeug zusätzlich zunächst mit einem TrafficSpike aus der SpeedCamera gestoppt und danach erfolgt durch die Polizei die Festnahme des Owner sowie die Beschlagnahmung des Car. Das Car mit dem Owner wird vom ParkingSpace entfernt und steht für die Auswahl nicht mehr zur Verfügung.

j) Bei Aufruf von **Report** wird eine Textdatei report.log mit folgenden Informationen erstellt [i] Anzahl der records, gruppiert nach manufacturer, [ii] Übersicht der records, aufsteigend sortiert nach timestamp, [iii] Übersicht der records, absteigend sortiert nach measuredSpeed, [iv] Übersicht mit minimaler, maximaler und durchschnittlicher Geschwindigkeitsüberschreitung. Die Auswertung ist weitestgehend mit Lambdas/Streams zu realisieren.

k) Bei Aufruf von **Export** werden alle records – aufsteigend sortiert nach timestamp – in eine CSV-Datei export.csv exportiert.

l) Bei Aufruf von **Shutdown** wird die SpeedCamera heruntergefahren (isShutdown = true).

m) Bei Aufruf von **Exit** wird die Applikation verlassen.

Es sind folgende **zukünftige Anforderungen** zu **berücksichtigen / prototypisch** zu **implementieren**.

1. Für die verschlüsselte Kommunikation zwischen VehicleRegistrationAuthority und MobileNetworkModule der SpeedCamera wird RSA genutzt.
2. Auf der IDCard der neuen Generation wird zusätzlich der fingerprint auf Basis von MD5 gespeichert. Wahlweise kann die pin oder der fingerprint für die Authentifizierung genutzt werden.
3. Nach dem Export wird die Datei export.csv mit AES zu export.enc verschlüsselt.
4. Das Car zu einem zur Fahndung ausgeschriebenen Owner wird mit einem der SpeedCamera zugeordneten Modul „EMP“ stillgelegt. Danach erfolgt die Festnahme und Beschlagnahmung.

Wichtige allgemeine Hinweise für die Bearbeitung

- Die **Bearbeitung** dieser Aufgabenstellung erfolgt **im Team mit maximal 6 Studierenden**.
- Bitte fügen Sie der Abgabe eine **readme.txt** mit der Zuordnung Aufgabe/Matrikelnummer bei.
- **Zielsetzungen** sind [i] zielorientiertes Anforderungsmanagement, [ii] weitestgehende Berücksichtigung der Design Prinzipien (SOLID), [iii] vollständige und strukturierte Modellierung und [iv] Implementierung und Test der Geschwindigkeitskontrolle.
- **Ggf. unvollständige und/oder mehrdeutige Formulierungen** in der Spezifikation sind durch sinnvolle Annahme zu **bitte korrigieren**. Über Feedback wäre ich sehr dankbar.
- **Verwendung** des **Style Guide** und **geeigneter englischer Begriffe**.
- Spezifikation **funktionale**r **Anforderungen** mit der **SOPHIST-Satzschablone** FunktionsMaster.
- Schematische Darstellung der Anwendung der **agile**n **Vorgehensweise** mit **Scrum**.
- Bei Bedarf kann das korrespondierende Diagramm in Teil-Diagramme aufgegliedert werden.
  In diesem Fall ist die **Konsistenz der Teil-Diagramme untereinander sicherzustellen**.
- Für die **Modellierung** wird **Visual Paradigm Community 17** und das **Template** genutzt.
  - Anwendungsfalldiagramm
  - Je Design Prinzip ein Klassendiagramm mit prägnantem Beispiel zu bad/good.
  - Klassendiagramm (unter Berücksichtigung der Design Prinzipien „SOLID“)
  - Aktivitätsdiagramm(e)
  - Sequenzdiagramm(e)
  - Zustandsdiagramm

  ► Bitte beachten Sie bei der Modellierung auf **syntaktische und semantische Korrektheit.**
- Bitte achten Sie bei der **Modellierung** auf ein **geordnetes Gesamtbild** (Look & Feel).
- Als **Entwicklung**sumgebung wird [i] **Java SE Development Kit 17.0.8**, [ii] **IntelliJ IDEA Community oder Ultimate 2023.2.3**, [iii] **gradle 8.4** und [iv] **GitHub** genutzt.
- **Implementierung**

  Durchführung der **Code Inspection** (Analyze ► Inspect Code) und die **Hinweise sind in Abhängigkeit von** dem **Language Level 17** weitestgehend **umzusetzen**.

  Es ist eine **geeignete** und **sinnvolle Paketstruktur** zu realisieren.

  Clean-Up und Formatierung des **Source Code**.
  (Code ► Reformat Code [Optimize imports, Rearrange entries, Cleanup code])

  Qualitätssicherung und **Testen** der Implementierung mit **JUnit 5**. Erstellung einer **Teststrategie** und **Nutzung leistungsfähige**r **Junit-/Mockito-Verfahren**.
  Testen Sie (zusätzlich) **einen Anwendungsfall Ihrer Wahl mit BDD (cucumber)**.
- **Bitte modellieren/implementieren** Sie **kollaborativ im Team**.
- **Erstellung** einer vollständigen und unverschlüsselten **7-Zip-Datei** (Kompressionsstärke: Ultra mit der Verzeichnissen **01_rm_diagram** und **02_implementation**.
- **Abgabetermin**: So., 05.11.2023

Studierender 01	Studierender 02 (oder 02/03)
• **Anforderungsmanagement** • Exemplarische Beschreibung der **agilen Vorgehensweise** mit **Scrum** **Modellierung | UML 01** • Design Prinzipien (SOLID) Je Design ein Klassendiagramm mit prägnantem Beispiel zu Bad/Good. • Klassendiagramm	**Modellierung | UML 02** • Anwendungsfalldiagramm • Aktivitätsdiagramm • Sequenzdiagramm • Zustandsdiagramm
Studierender 03 (oder 04/05)	**Studierender 04 (oder 06)**
Implementierung auf Basis dem finalen Klassendiagramm unter Berücksichtigung der zukünftigen Anforderungen (SOLID).	• Qualitätssicherung und Testmanagement für zentrale Anwendungsfälle. • Realisierung von einem Test mit BDD für einen Anwendungsfall nach Wahl.