2021 FALL COEN6311 Assignment 1 Report

Jun Huang
40168167
Email: youyinnn@foxmail.com

I. Deliverable Specification

D1-User Stories and Use Cases

According to the project description, the user stories of the system can be listed as follow:

1) The user wants to track the GPS data of the bike which represent on a digital map, so he can know the real-time location of it.
2) The user wants to know the speed of the bike, so he can know if he can catch the moving stolen bike.
3) The user wants to set a "Parked" state to the bike, so the chip(micro-controller) would treat the any swing as a threat of stealing.
4) The user wants to set the "Unparked" state to the bike, so the chip would not treat the swing perform by the user as a threat of stealing.
5) The user wants to know if the bike was swinging or not, so he can know if someone was attempting to move his bike or to break his lock.
6) The user wants to get the real-time notifications which described above, so he can know the state of the bike immediately.

The use cases diagram is shown in Fig. 1.

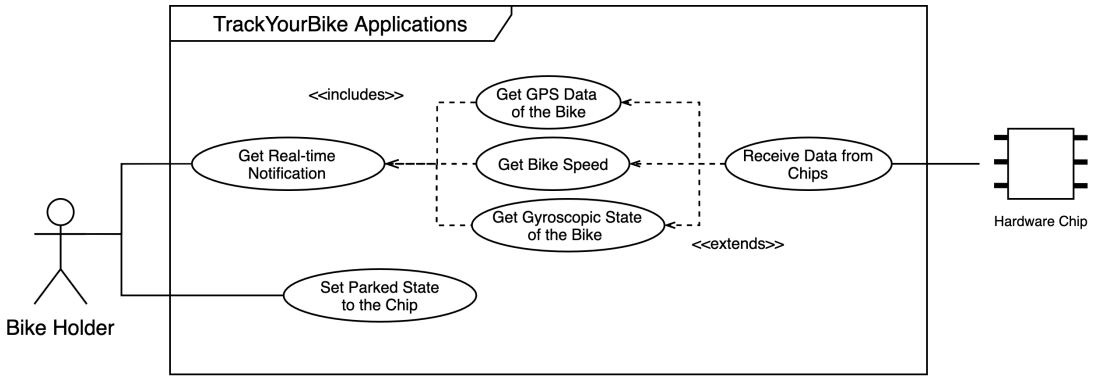


Fig. 1. System Use Cases

D2-Software Requirement Specification

Based on the user scenairo and the use cases, the software shall have the following software requirement specifications and their sub-requirements.

To fulfill the described software product, the overall product should contains at least 3 software systems:

- *Software on the Tracker Chip*: which will be installed in the hardware chip to perform tracking features and send sensor data to the remote server.
- *Cloud Server System*: which will be deployed on the cloud server and receive data from users' chips then forwarding those data to users' applications.
- *Client Application*: which will be installed on users' mobile devices then receive and display the data or notification from the server, and also will send instructions to the corresponding chips.

This report will focus on the *Software on the Tracker Chip* but also present part of the *Cloud Server System* which is just for supplementary notes. The *Client Application* will not be presented. Notice the SRS of the cloud system will be marked as *S-SRS* for distinction.

More information of the system context architecture is shown in Fig. 2.

SRS for the Software on the Tracker Chip

Total of 4 SRSs are defined for the *Software on the Tracker Chip*. In the scope of the assignment report, SRS 1, 2 and 3 are expanded with sub-requirements.

- *SRS 1.* The chip system shall receive the raw data sent from other sensors and adapt those raw data into more structured data and send all structured data to the cloud server.
  - *1.1 Data Process Module:* this module shall handle the message construction part. The module shall collect the raw data from 3 hardware components and build them into a json format data.
  - *1.2 Cloud Server Communication Module:* this module shall handle the communication part. The module shall be able to receive message from the cloud server and send json data described above to the server.
- *SRS 2.* The chip system shall be able to receive a state data sent from the cloud server which represents a state of *Parked* and *Unparked* and store this state into the memory of the micro-controller.
  - *2.1 State Management Module:* this module shall set the states described above into the chip.

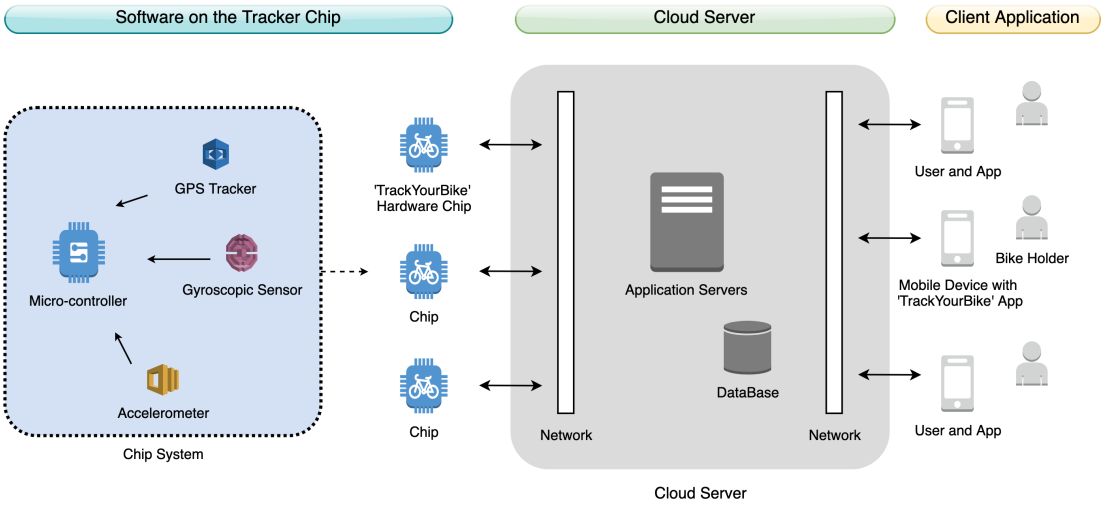


Fig. 2. System Context Architecture

- *SRS 3.* The chip system shall be able to detect whether the bike is under the risk of stealing based on the gyroscopic data.
  3.1 Risk Detection Module: this module shall be able to estimate the risk level based on the gyroscopic data.
- *SRS 4.* The chip system shall sent a warning message to the cloud server to inform the user that their bike is under the risk of stealing when the state is at *Parked*.

SRS for the Cloud Server System

- *S-SRS 1.* The server system shall receive http request and maintain socket connection with multiple clent-end.
- *S-SRS 2.* The system shall maintain the relationship between client-ends or hardware chips.
- *S-SRS 3.* The system shall receive chips data instantly and organize them into database, and those data shall be analysed and pushed to the corresponding clent-end device.
- *S-SRS 4.* The system shall receive 'Parked' and 'Unparked' commands sent from a certain client-end and associate them with the ressorted hardware chip instance.

D3-System Design

System Architectures

Fig. 3(a) depicts the *Software on the Tracker Chip*'s top-down layer system architecture. The top layer is dependent on the bottom layer in objects with vertical layer positions. The grey box indicates that the objects do not have a hierarchy or layer relationship, they are on the same architectural level.

Then a same top-down layer system architecture of the *Cloud Server System* is shown in Fig 3(b).

Activitiy Diagram of Some Key Processes on the Software on the Tracker Chip

The activity diagrems are shown in Fig. 4.

D4-Define An Incremental Process

The defined incremental process is shon in Fig. 5. The process is consist of the techniques like "Incremental Planning" and "Test First Development" from the *Extreme Programming*.

To perform the incremental process, a loop starts at step 4 and ends with step 7 when no unfinished tasks are left on the storyboard. Every task will start from defining the test cases and then developing and validating.

The SRS 2 is chosen to practice the incremental process with XP methodologies.

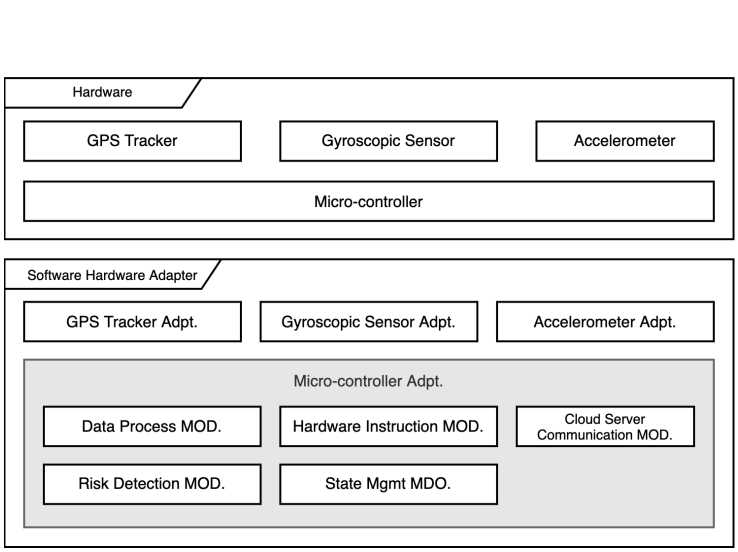
D5-Practice with the Extreme Programming

Story Cards

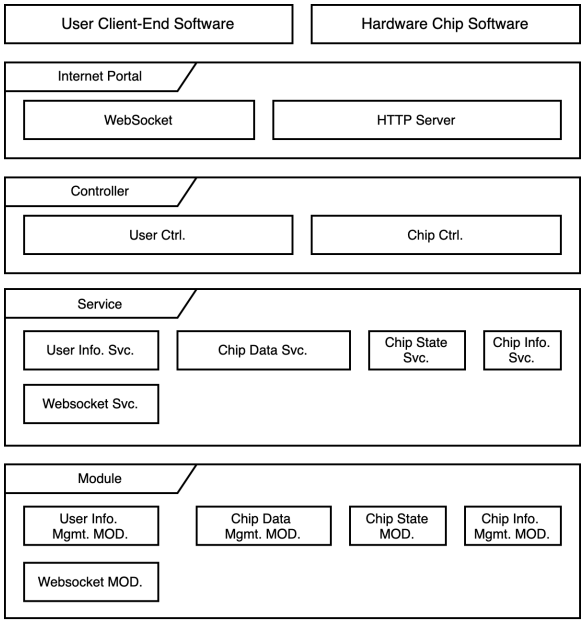
Following the *Incremental Planning* technique, SRS 2 can be partitioned as several story cards(tasks) which are shown in Table I.

TABLE I
STORY CARDS(TASKS) OF SRS 2

No.Title	**0. Mock Raw Data**	**1. Data Class Definition**	**2. Build Raw Data into Json Data**
Description	Mock the raw data which sent from the hareware sensors for unit test verification	Define classes which represent the raw data	A module to construct the raw data into the Json Data

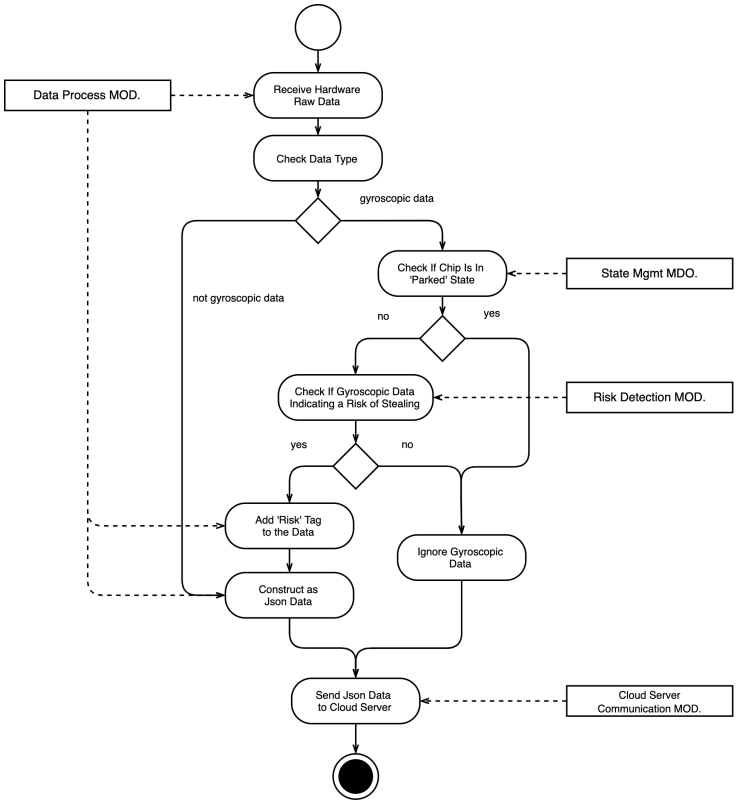


(a) Architecture of the *Software on the Tracker Chip*

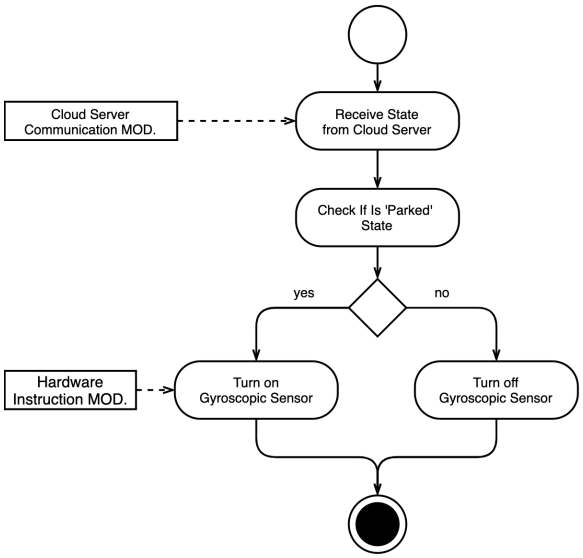


(b) Architecture of the *Cloud Server System*

Fig. 3. System Architectures: (a) (b)



(a) Activitiy Diagram of SRS 1, 3, and 4



(b) Activitiy Diagram of SRS 2

Fig. 4. Activity Diagrams of the SRS

Finish the Task

Task 1 is chosen to be presented in this report. However, mocked sensor raw data is required for the unit test. With the mock data, the console of the IDE will represent the display device that the client will use, and all raw data will be reconstructed to json data. This implies that the task result will be validated manually by observing the console.

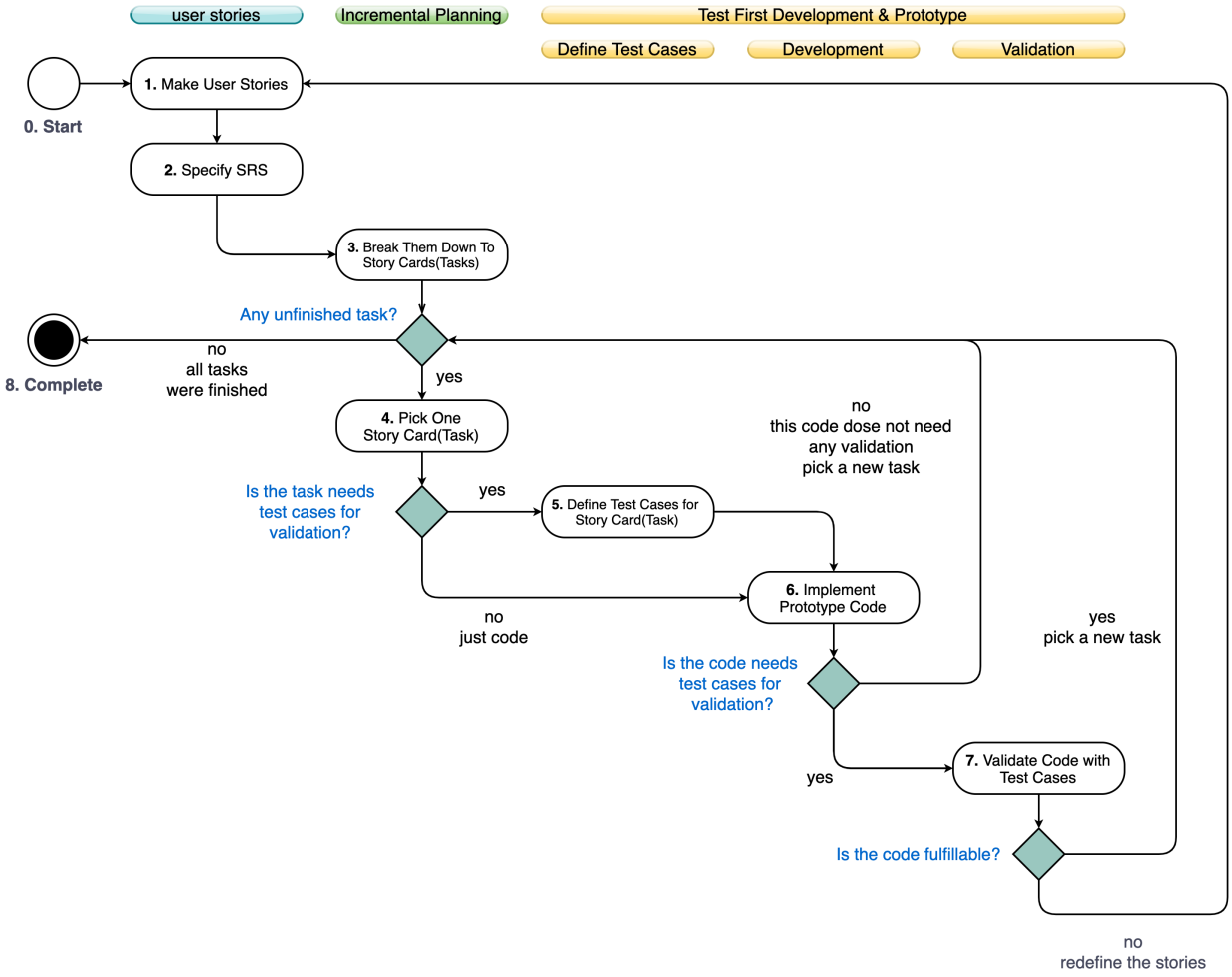


Fig. 5. Incremental Process

Task 0

Task 0 started first. As can be seen in Fig. 6, the code will generate one raw data record of the GPS tracker. So as the raw data of accelerometer and the gyroscopic sensor. The code will be located in "src/test/java/io/github/youyinnn/bo/chip/RawDataMocker.java".

```java
/**
 * Top latitude and longitude of Montreal.
 */
private static final String TOP_LATITUDE = "45";
private static final String TOP_LONGITUDE = "-73";
private static final Random RANDOM = new Random();
private static final int GPS_MANTISSA_LENGTH = 15;

/**
 * Mocking the mantissa with the length of 'mantissaLength'
 */
private static String getMantissa(final int mantissaLength) {
    StringJoiner sj = new StringJoiner("");
    for (int i = 0; i < mantissaLength; i++) {
        final int random = RANDOM.nextInt(10);
        sj.add(random + "");
    }
    return sj.toString();
}

private static String mockGpsData() {
    // mocking gps raw data like:
    // -73.807329854301666 45.206528409572911
    return TOP_LONGITUDE + "." + getMantissa(GPS_MANTISSA_LENGTH) + " "
            + TOP_LATITUDE + "." + getMantissa(GPS_MANTISSA_LENGTH) + (System.lineSeparator());
}
```

Fig. 6. Test Cases of Generating Raw Data of GPS Tracker

In the first loop, the raw data did not contain the timestamp, so the test cases are not fulfilled since those data are displayed in a real-time situation. As the result, a second loop was started, and the incremental code is presented in Fig. 7.

```
private static String mockGpsDataSet(int bound) {
    StringJoiner sj = new StringJoiner("");
    int setSize = RANDOM.nextInt(bound);
    if (setSize == 0)
        setSize++;
    for (int i = 0; i < setSize; i++) {
        sj.add(String.valueOf(System.currentTimeMillis())); // incremental part in loop 2: adding timestamp
        sj.add(" ");
        sj.add(mockGpsData());
        try {
            Thread.sleep(RANDOM.nextInt(30 * 100)); // incremental part in loop 2: mocking the time delay
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    return sj.toString();
}
```

Fig. 7. Revised Code which Generating Timestamp before the Raw Data

The revised sample of the raw data is shown in 9. Those data files are located in "src/main/resources/data_samples".

Task 1

- **Test Cases:** The test cases for task 1 can be defined as Fig. 10 is shown. Its location is "src/test/java/io/github/youyinnn/module/chip/ChipDataProcessorImplTest.java".
- **Code Implementation:** After test cases defined, then the implementation of the *ChipDataProcessor* was implemented which is *ChipDataProcessorImpl* and is shown in Fig. 11. The corresponding class diagrams of the data objects are shown in Fig. 12. And their implementations are located in "src/main/java/io/github/youyinnn/bo/chip".
- **Validation:** After all the implementations were done, the test units were executed and they were all passed(shown in Fig. 8). This means the task 1 was completed and the loop for this task was finished.

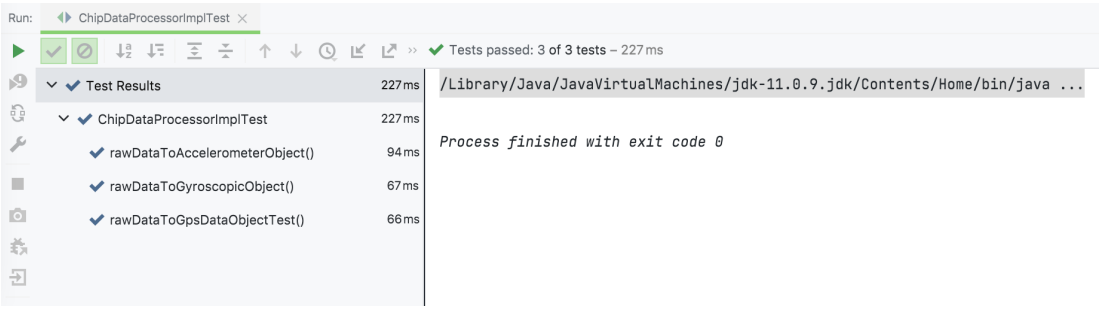


Fig. 8. Result of the Unit Test of Task 1

gps.data

src > main > resources > data_samples > gps.data

```
1635110960954 -73.807329854301666 45.206528409572911
1635110961369 -73.950945663165644 45.315112702200464
1635110961812 -73.387340436141720 45.301724240274980
1635110963257 -73.479766745422494 45.238150943043081
1635110964708 -73.016545072909787 45.423596042616563
1635110965506 -73.218200259456532 45.805367463189291
1635110967806 -73.267884346623038 45.746151335516208
1635110970155 -73.475559473741200 45.092076046283034
1635110971714 -73.183133947960566 45.597179811631200
1635110973808 -73.308533801826849 45.266337104259073
1635110974810 -73.678503428590409 45.944767622772943
1635110974974 -73.093095720358707 45.545631903407639
1635110976575 -73.762649711594136 45.940865816653414
1635110977262 -73.104775628826664 45.986359914093251
```

(a) Raw Data of GPS Tracker

accelerometer.data

src > main > resources > data_samples > accelerometer.data

```
1635111020447 7.97
1635111020506 2.69
1635111022492 5.67
1635111022923 1.03
1635111024520 7.81
1635111024599 3.72
1635111027497 7.82
1635111028588 6.06
1635111028958 8.86
1635111031418 6.41
1635111031857 5.77
1635111031929 7.06
1635111033436 3.45
1635111035523 8.19
```

(b) Raw Data of Accelerometer

gyroscopic.data

src > main > resources > data_samples > gyroscopic.data

```
1635111108764 173.422429317704999 -57.622175742617439 -164.922731045570664
1635111111501 -146.141568630228786 -37.368460557644922 172.250725071942288
1635111114491 -62.435259142557761 -66.709094116901181 152.702204260420808
1635111117009 113.938688868953972 -121.517805122535281 -118.403696297241515
1635111117058 -66.976041088972792 -1.020420886482615 -161.263339571027880
1635111118209 89.077600134542667 -111.337635384246202 177.403991725882625
1635111118410 130.114382897866307 -1.004010355448198 -132.367252463724879
1635111119916 11.478060434716485 151.559688944396761 -108.592699739776537
1635111122759 -109.692740805245309 105.337183330198528 -115.654257012675919
1635111124338 94.094895648323051 144.615282579180386 -88.235698272843117
1635111126840 -72.037479048630396 -87.514961358385346 109.048578281152968
1635111127636 -71.926686444044314 33.763252393958773 -31.422299450702646
1635111128888 -74.861801575621608 -25.112377129798619 162.793881473309825
1635111130690 48.731664466189987 60.302343868582815 -3.013311926519001
```

(c) Raw Data of Gyroscopic Sensor

Fig. 9. Mocked Raw Data

```java
public interface ChipDataProcessor {

    GpsData rawDataToGpsDataObject(String chipNo, String rawDataString);

    AccelerometerData rawDataToAccelerometerObject(String chipNo, String rawDataString);

    GyroscopicData rawDataToGyroscopicObject(String chipNo, String rawDataString);

    String toOuterMessage();

}
```

(a) *ChipDataProcessor* Interface

```java
private static final String CHIP_NO = "qjjfapdj8810296690123";

private final static ClassLoader classLoader = ChipData.class.getClassLoader();

private static String gpsDataString;
private static String gyroscopicDataString;
private static String accelerometerDataString;

private static final Gson gson = new Gson();
// the implementation of the interface ChipDataProcessor
private static final ChipDataProcessor chipDataProcessor = new ChipDataProcessorImpl();

// get mocked raw data as string
private static String getFileString(final String filePath) throws IOException {
    final URL url = classLoader.getResource(filePath);
    assert url != null;
    final File file = new File(url.getPath());
    return Files.readString(file.toPath());
}

// retrieve all raw data from files before all test cases was started
@BeforeAll
public static void beforeAll() throws IOException {
    gpsDataString = getFileString("data_samples/gps.data");
    gyroscopicDataString = getFileString("data_samples/gyroscopic.data");
    accelerometerDataString = getFileString("data_samples/accelerometer.data");
}
```

(b) Preparing the Raw Data

```java
private void compareGpsDataJson(String rawData, GpsData gpsData) {
    final String[] part = rawData.split(" ");
    // build json data manually
    String sj = "{\"timestamp\":" + part[0] + "," +
            "\"chipNo\":\"" + CHIP_NO + "\"," +
            "\"lon\":" + part[1] + "," +
            "\"lat\":" + part[2] +"}";

    // transfer manually built json data string into JsonElement
    final JsonElement jsonElementFromRawData
            = JsonParser.parseString(sj);

    // transfer gpsData object which read from the data file into JsonElement
    final JsonElement jsonElementFromProcessor
            = JsonParser.parseString(gson.toJson(gpsData));

    // assert they were the same
    Assertions.assertEquals(
            jsonElementFromRawData,
            jsonElementFromProcessor
    );
}
```

(c) Assertion of the Test Unit

```java
/**
 * Test unit for validating the GPS data
 */
@Test
void rawDataToGpsDataObjectTest() {
    final String[] split = gpsDataString.split(System.lineSeparator());

    for (String one : split) {
        final GpsData gpsData
                = chipDataProcessor.rawDataToGpsDataObject(CHIP_NO, one);
        // compare 1 by 1
        compareGpsDataJson(one, gpsData);
    }
}
```

(d) Test Unit

Fig. 10. Test Case Code of Task 1

```java
public class ChipDataProcessorImpl implements ChipDataProcessor {

    @Override
    public GpsData rawDataToGpsDataObject(String chipNo, String rawDataString) {
        final String[] part = rawDataString.split(" ");
        return new GpsData(chipNo, Long.valueOf(part[0]),
                Double.valueOf(part[1]), Double.valueOf(part[2]));
    }

    @Override
    public AccelerometerData rawDataToAccelerometerObject(String chipNo, String rawDataString) {
        final String[] part = rawDataString.split(" ");
        return new AccelerometerData(chipNo, Long.valueOf(part[0]),
                Double.valueOf(part[1]));
    }

    @Override
    public GyroscopicData rawDataToGyroscopicObject(String chipNo, String rawDataString) {
        final String[] part = rawDataString.split(" ");
        return new GyroscopicData(chipNo, Long.valueOf(part[0]),
                Double.valueOf(part[1]),
                Double.valueOf(part[2]),
                Double.valueOf(part[3]));
    }

    @Override
    public String toOuterMessage() {
        return null;
    }
}
```

Fig. 11. Implementation of the Interface *ChipDataProcessor*

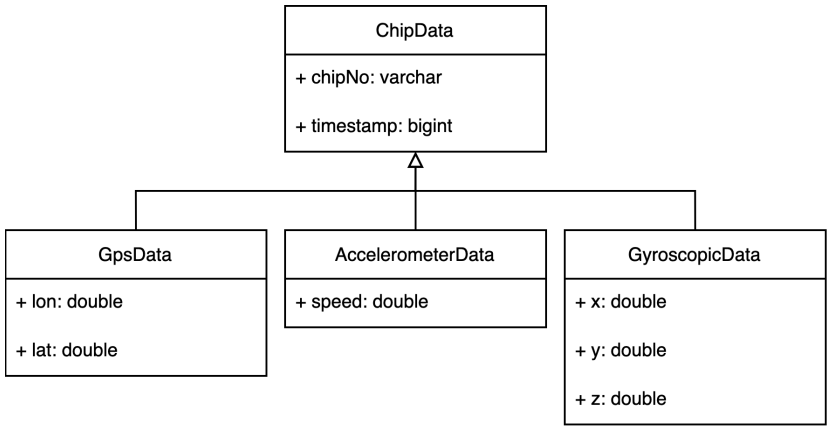


Fig. 12. Class Diagram of the Data Objects