FINAL DELIVERABLE

Vision

For this project we will be designing a piece of wearable technology to track and report the users vitals and health. The system has a few hardware requirements such as it needs to be wearable as some sort of band or watch. Second, it needs to have the necessary sensors to keep track of the users vitals such as their heart rate and oxygen levels. It should also have the correct tools to sense things such as movement and proximity to the ground (gyroscope). Along with the base hardware requirements there are also needs to be internet and bluetooth capable. This will require the correct hardware and software to be successfully implemented. The user interface also needs to be simple and efficient and allow the user to do things such as manipulate settings, keep track/view health and goals, as well, the UI needs to allow the user to do basic things such as view time, date, weather, calendar. If all of these can be successfully implemented we should be able to design and produce a competitive product for the wearable technology world.

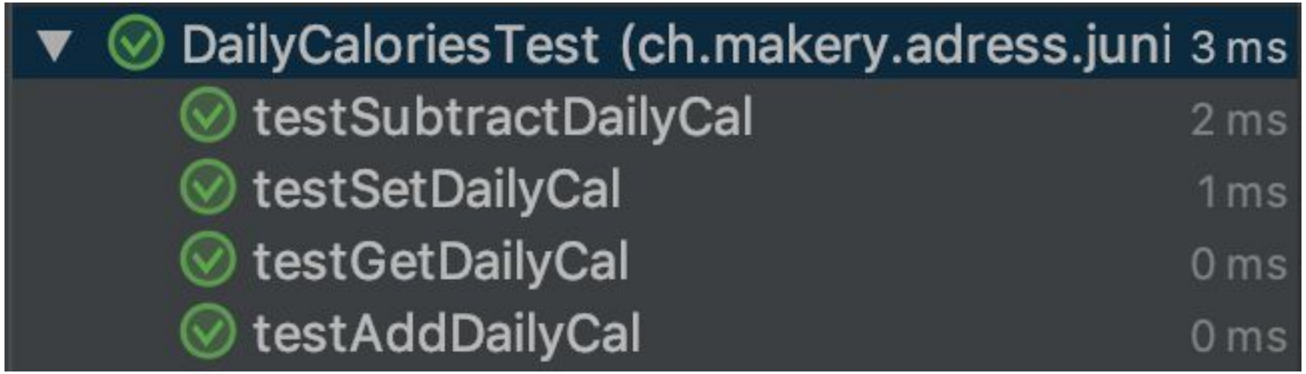
Requirements

The application must:

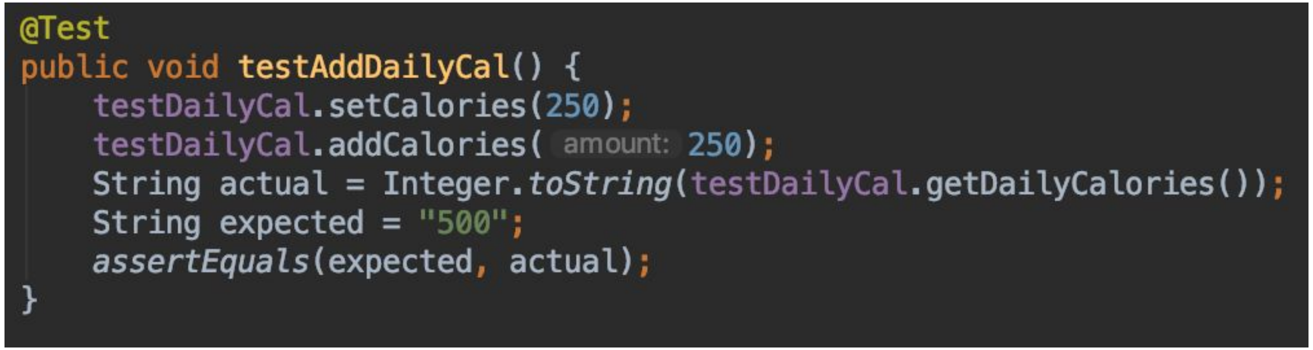
1. Record and report accurate health data captured from the sensor
2. Have a UI that is easy and intuitive to navigate
3. Provide options to the user to update their health information
4. Have the ability to display the current time

Analysis

DailyCaloriesTest

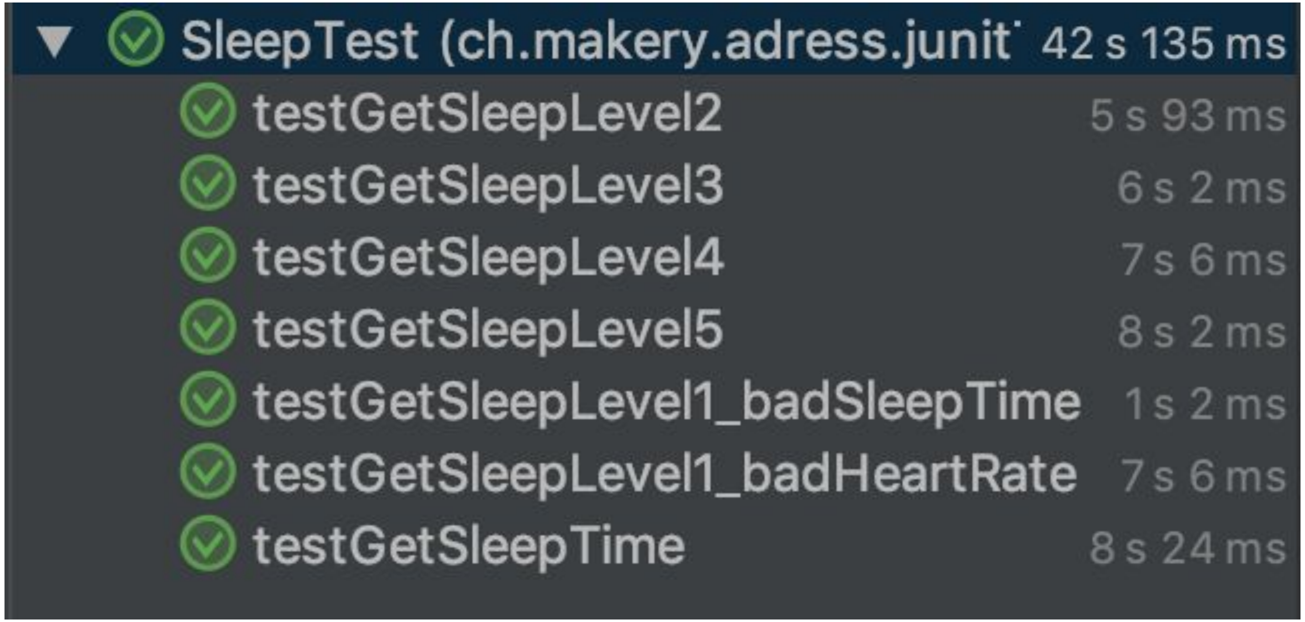

```
▼ ✓ DailyCaloriesTest (ch.makery.adress.juni  3 ms
      ✓ testSubtractDailyCal                    2 ms
      ✓ testSetDailyCal                         1 ms
      ✓ testGetDailyCal                         0 ms
      ✓ testAddDailyCal                         0 ms
```

Basic JUnit Test


```
@Test
public void testAddDailyCal() {
    testDailyCal.setCalories(250);
    testDailyCal.addCalories( amount: 250);
    String actual = Integer.toString(testDailyCal.getDailyCalories());
    String expected = "500";
    assertEquals(expected, actual);
}
```

Analysis

SleepTest

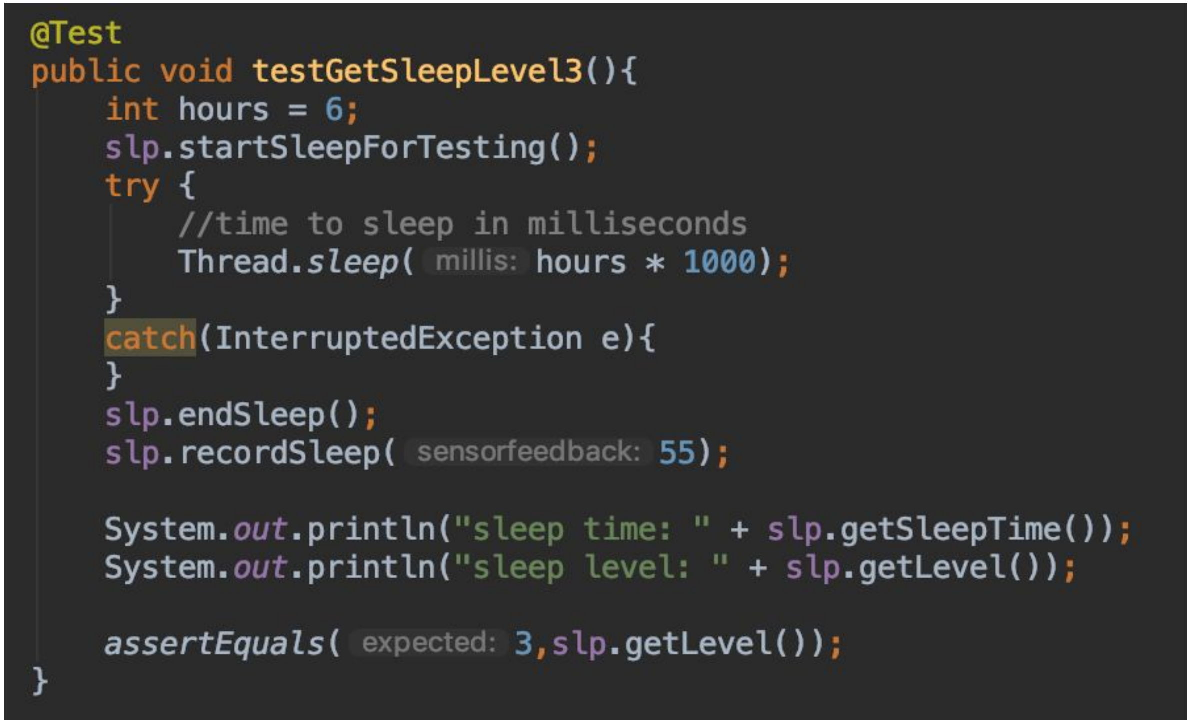


```
▼ ✓ SleepTest (ch.makery.adress.junit    42 s 135 ms
      ✓ testGetSleepLevel2                    5 s 93 ms
      ✓ testGetSleepLevel3                     6 s 2 ms
      ✓ testGetSleepLevel4                     7 s 6 ms
      ✓ testGetSleepLevel5                     8 s 2 ms
      ✓ testGetSleepLevel1_badSleepTime        1 s 2 ms
      ✓ testGetSleepLevel1_badHeartRate        7 s 6 ms
      ✓ testGetSleepTime                      8 s 24 ms
```

Analysis

SleepTest

- More Complex Test



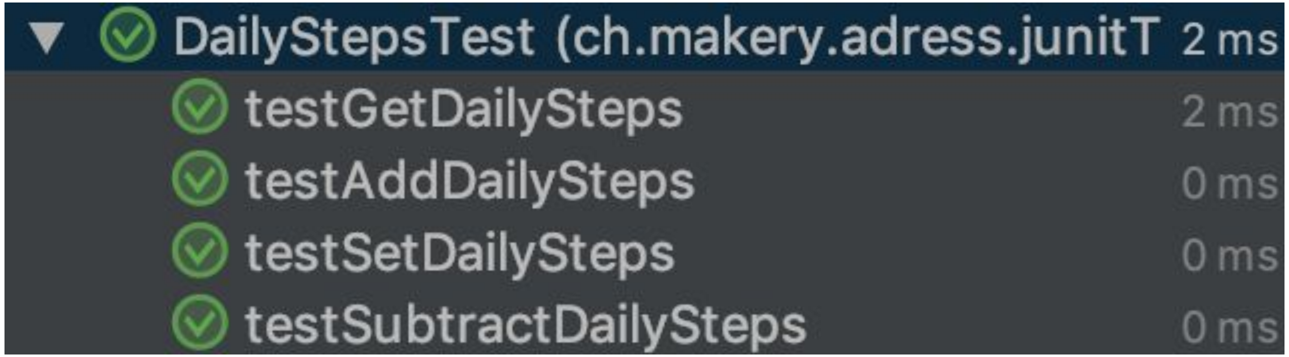
```
@Test
public void testGetSleepLevel3(){
    int hours = 6;
    slp.startSleepForTesting();
    try {
        //time to sleep in milliseconds
        Thread.sleep( millis: hours * 1000);
    }
    catch(InterruptedException e){
    }
    slp.endSleep();
    slp.recordSleep( sensorfeedback: 55);

    System.out.println("sleep time: " + slp.getSleepTime());
    System.out.println("sleep level: " + slp.getLevel());

    assertEquals( expected: 3,slp.getLevel());
}
```

Analysis

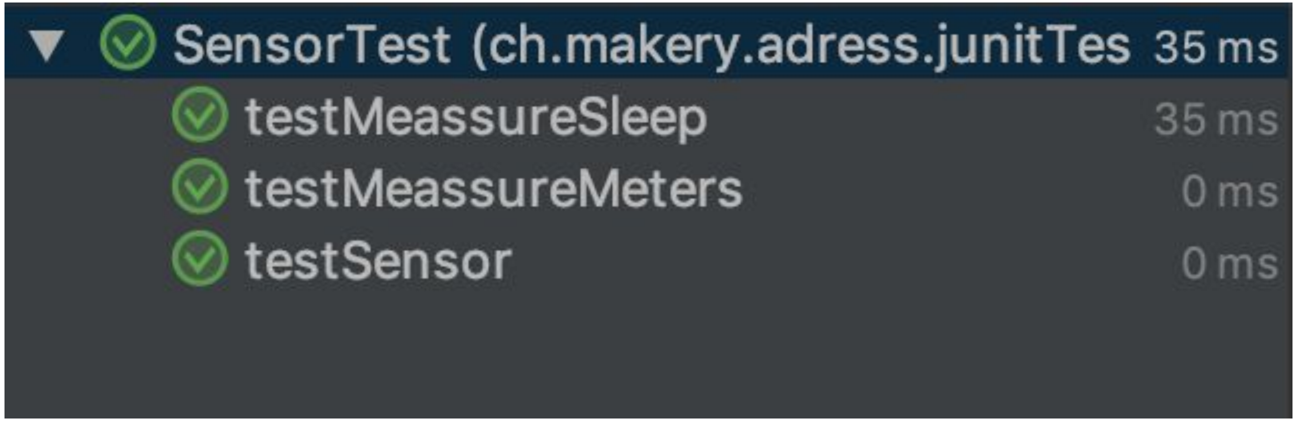
DailyStepsTest



```
▼ ✓ DailyStepsTest (ch.makery.adress.junitT 2 ms
      ✓ testGetDailySteps                     2 ms
      ✓ testAddDailySteps                     0 ms
      ✓ testSetDailySteps                     0 ms
      ✓ testSubtractDailySteps                0 ms
```

Analysis

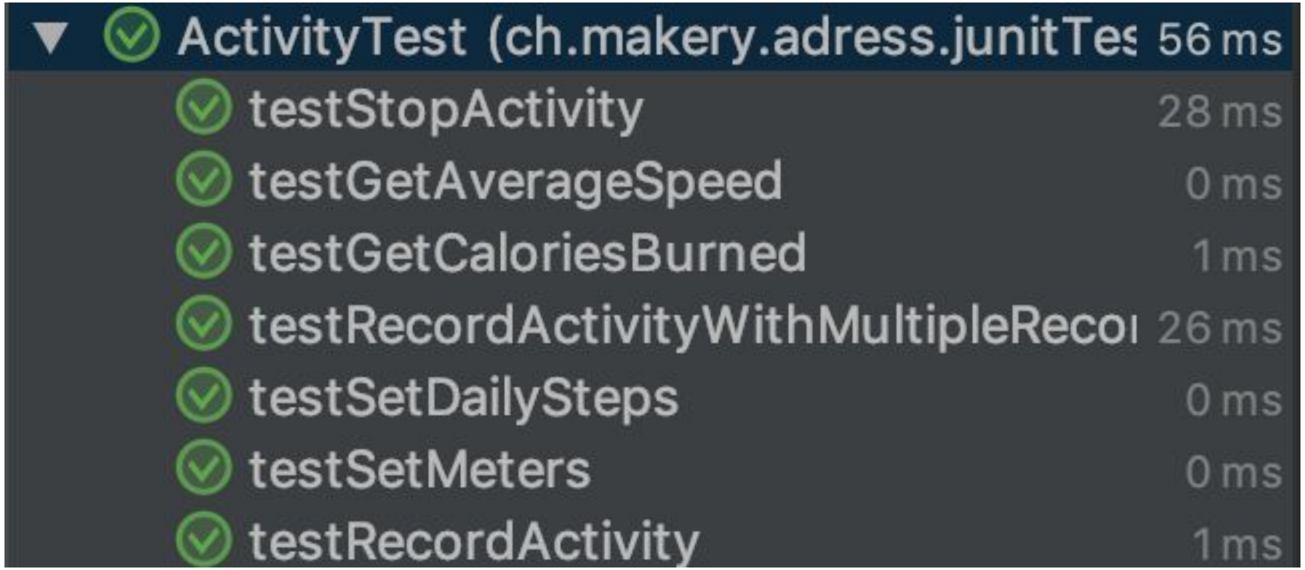
SensorTest



```
▼ ✓ SensorTest (ch.makery.adress.junitTes 35 ms
    ✓ testMeassureSleep                   35 ms
    ✓ testMeassureMeters                   0 ms
    ✓ testSensor                           0 ms
```

Analysis

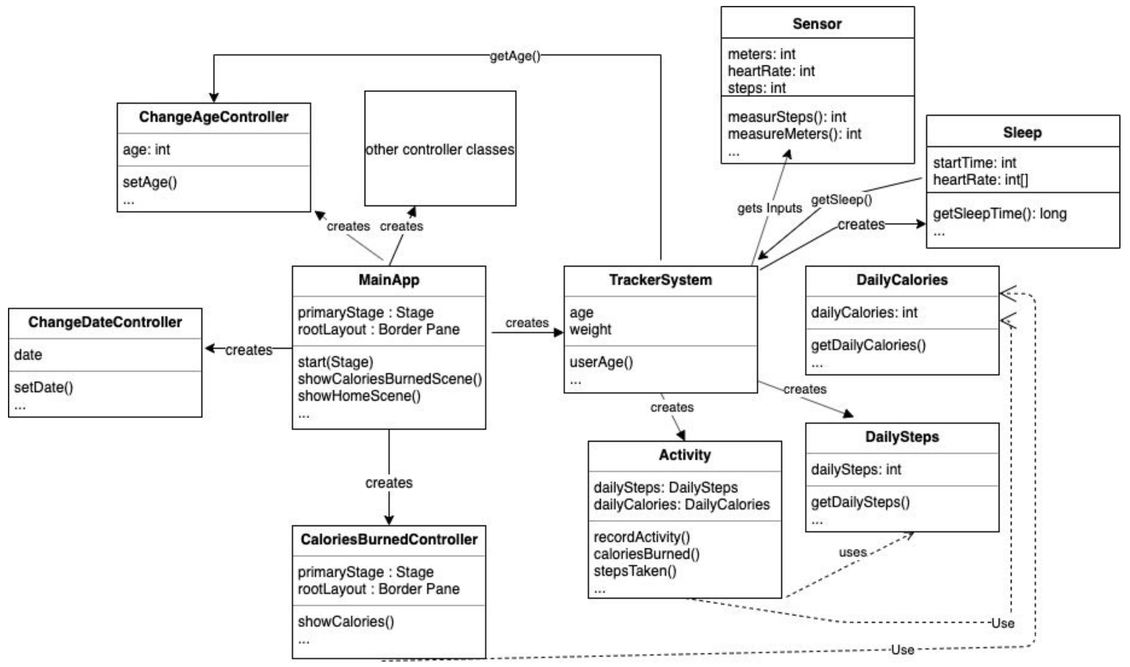
ActivityTest



```
▼ ✓ ActivityTest (ch.makery.adress.junitTes 56 ms
      ✓ testStopActivity                     28 ms
      ✓ testGetAverageSpeed                   0 ms
      ✓ testGetCaloriesBurned                 1 ms
      ✓ testRecordActivityWithMultipleRecoı  26 ms
      ✓ testSetDailySteps                     0 ms
      ✓ testSetMeters                         0 ms
      ✓ testRecordActivity                    1 ms
```

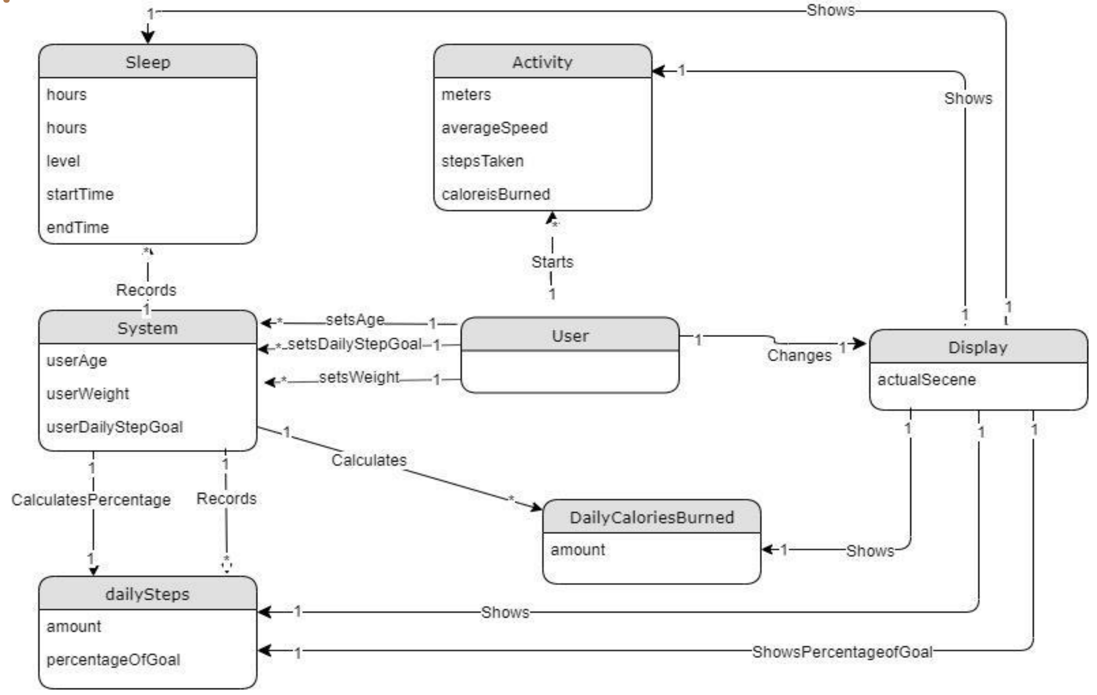
System Design

Class Diagram:



System Design

Domain Model:



Objects

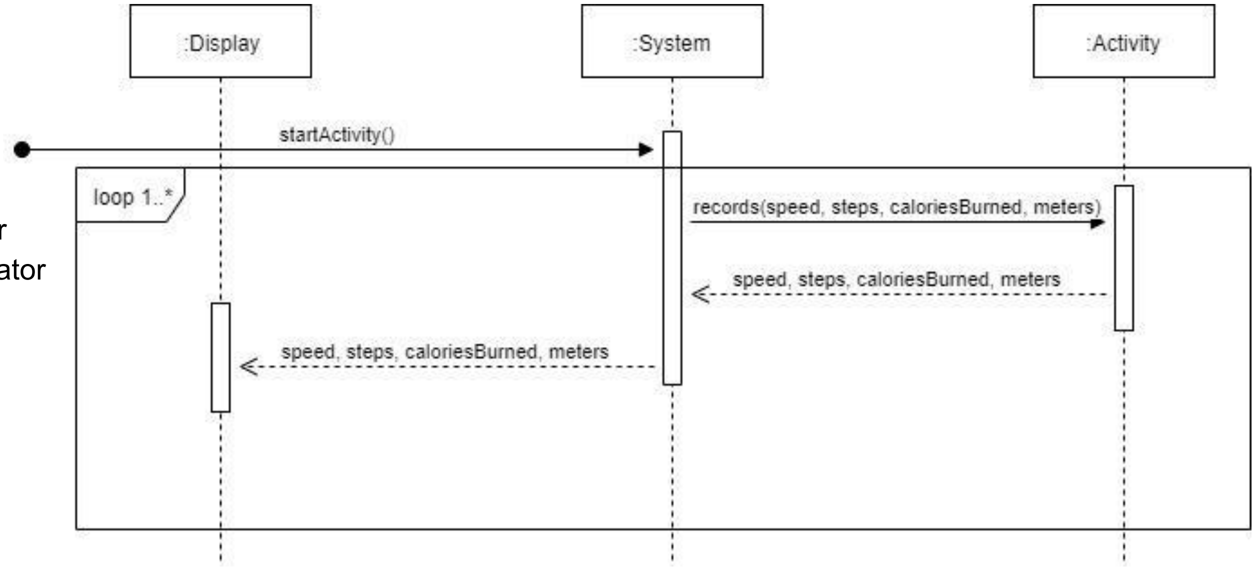
Class	Responsibility	GoF pattern
MainApp	Starts the program, creates controllers and System, and controls the scene flow	Factory and adapter
Activity	Saves data about activity	Observer and strategy
DailyCalorie	Saves calorie data	Observer
DailySteps	Saves steps data	Observer
ColoriesBurnedController	Controls the calorie burned scene	Facade
TrackerSystem	manages activity and user data	Factory, observer, and facade
ChangeAgeController	Controls change age scene	Facade
ChangeDateController	Controls change date scene	Facade
Sleep	Saves the sleep information	Observer

Objects

Class	Responsibility	GRASP pattern
MainApp	Starts the program, creates controllers and System, and controls the scene flow	Creator, controller, and pure fabrication
Activity	Saves data about activity	Information expert, low coupling
DailyCalorie	Saves calorie data	Information expert
DailySteps	Saves steps data	Information expert
ColoriesBurnedController	Controls the calorie burned scene	Controller
System	manages activity and user data	Information expert and pure fabrication
ChangeAgeController	Controls change age scene	controller
ChangeDateController	Controls change date scene	controller
Sleep	Saves the sleep information	Information expert

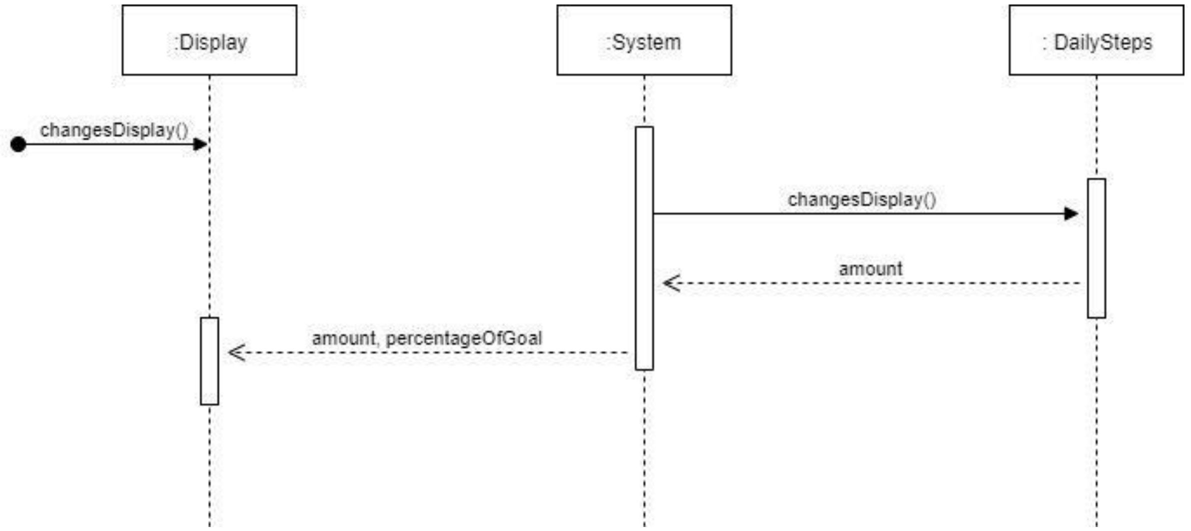
Sequence Diagrams

- Recording activity
    - :system is the controller
    - :activity acts as the creator
    - High cohesion

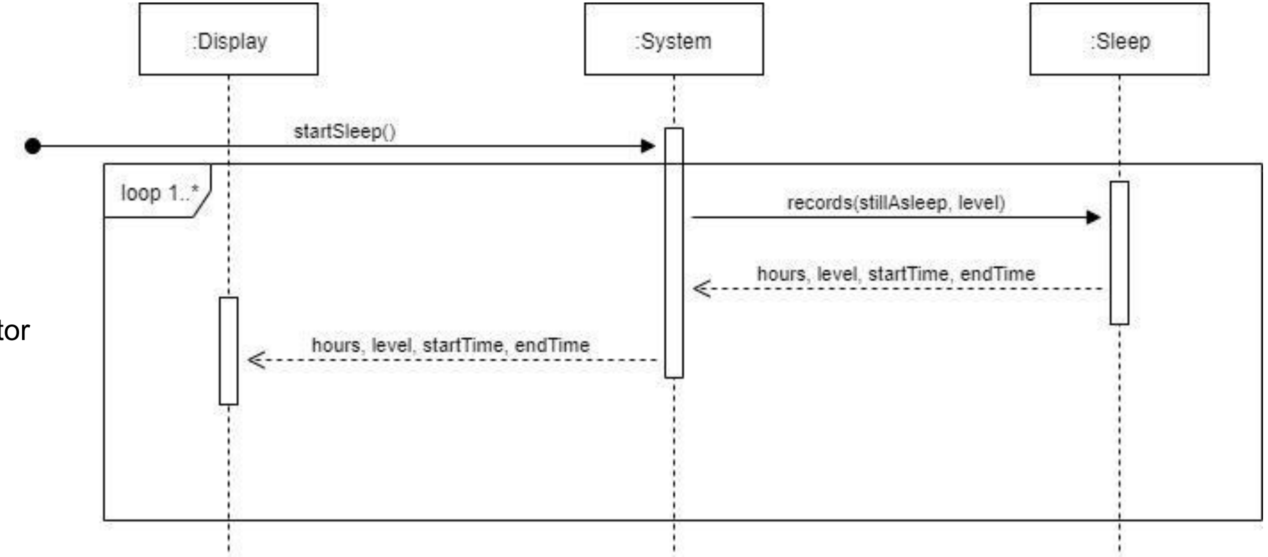

:Display
:System
:Activity
startActivity()
loop 1..*
records(speed, steps, caloriesBurned, meters)
speed, steps, caloriesBurned, meters
speed, steps, caloriesBurned, meters

Sequence Diagrams

- See daily steps
  - :system is the controller
  - :dailySteps acts as the information expert
  - High cohesion



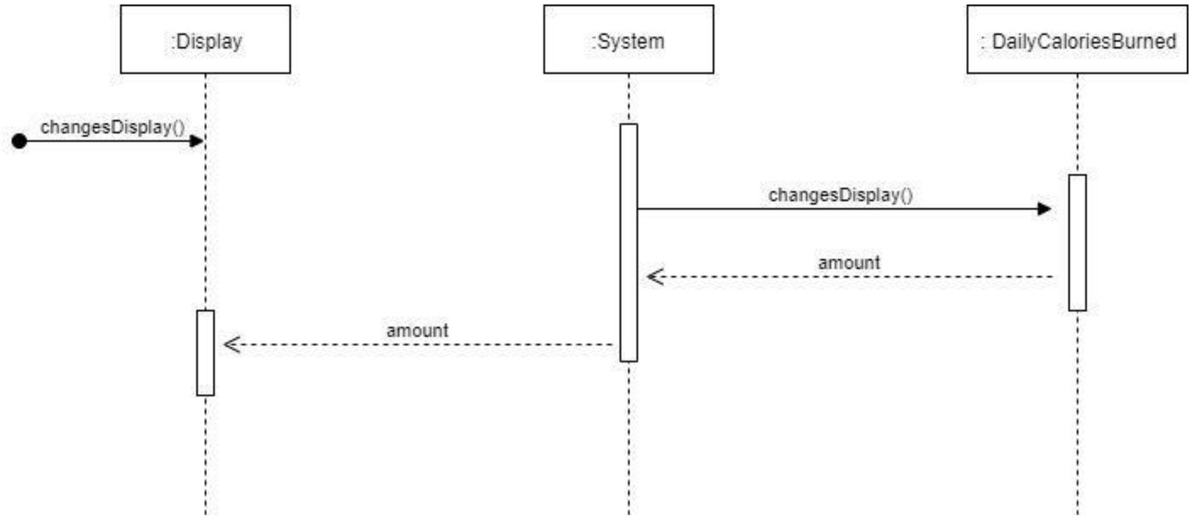
Sequence Diagrams


:Display
:System
:Sleep
startSleep()
loop 1..*
records(stillAsleep, level)
hours, level, startTime, endTime
hours, level, startTime, endTime

- Start sleep
  - :system is the controller
  - :activity acts as the creator
  - high cohesion

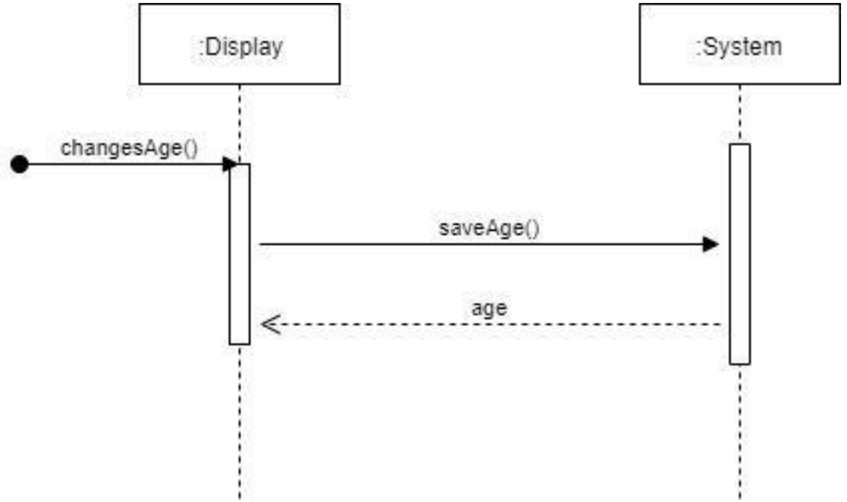
Sequence Diagrams

- See Calories Burned
    - :system is the controller
    - :dailyCaloriesBurned acts as the information expert
    - high cohesion



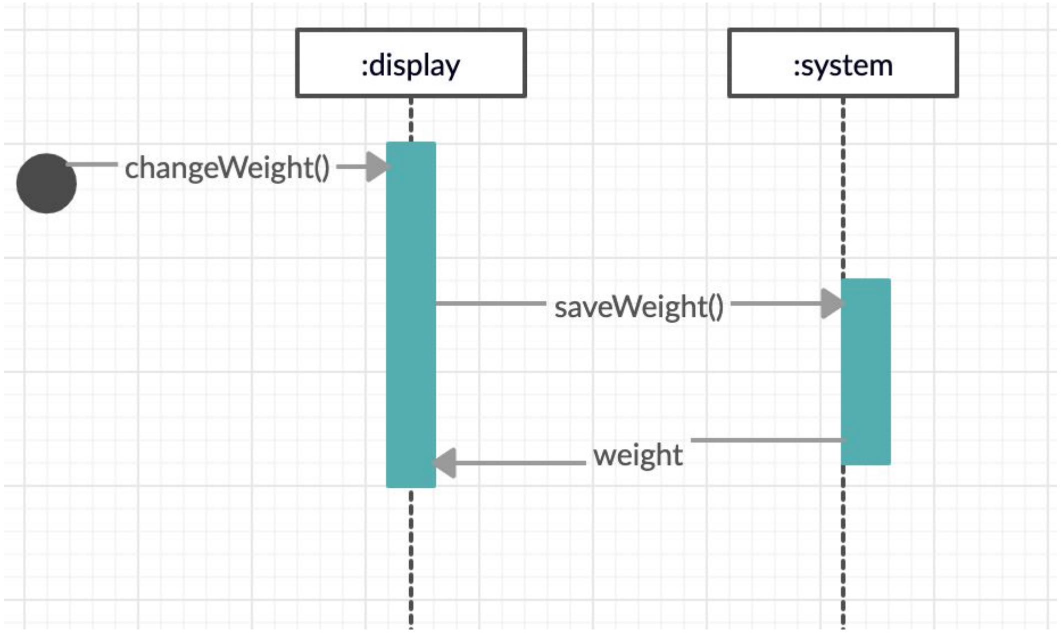
Sequence Diagrams

- Change age
    - :display acts as controller
    - :System act as the creator
    - high cohesion
    - Low coupling



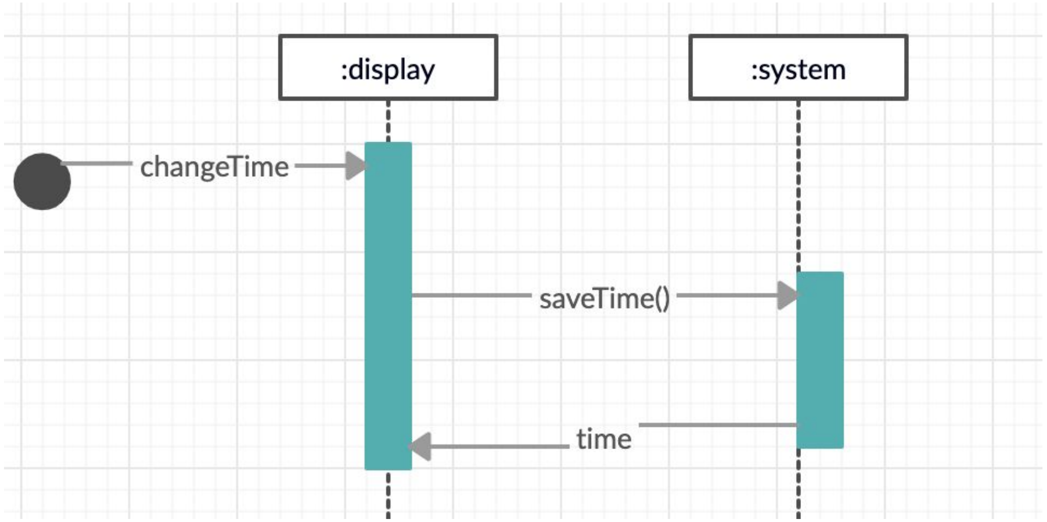
Sequence Diagram

- Change weight
  - :display acts as controller
  - :System act as the creator
  - high cohesion
  - Low coupling



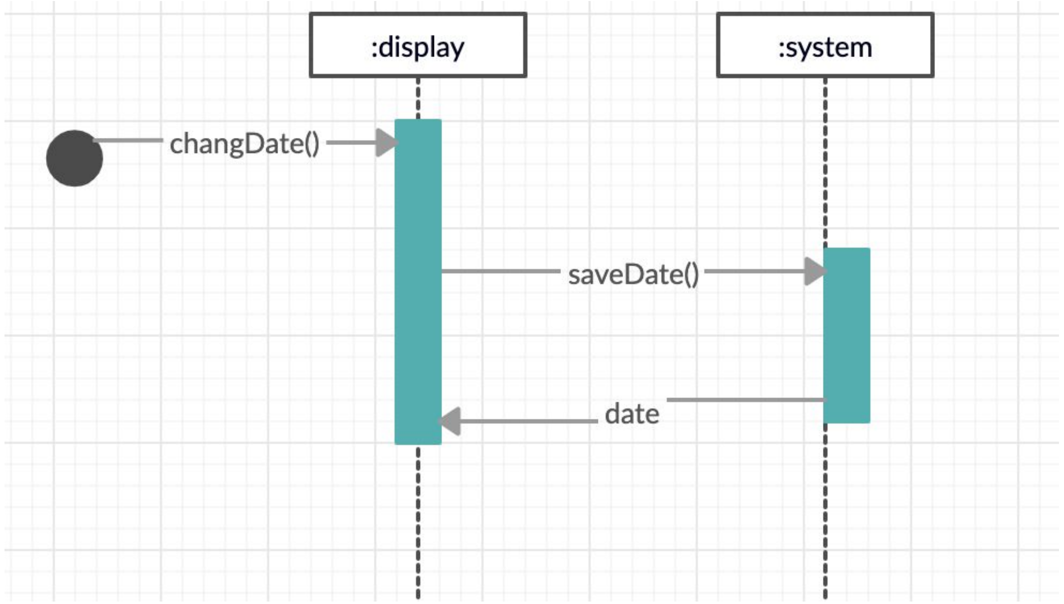
Sequence Diagram

- Change time
  - :display acts as controller
  - :System act as the creator
  - high cohesion
  - Low coupling



Sequence Diagram

- Change date
  - :display acts as controller
  - :System act as the creator
  - high cohesion
  - Low coupling



DEMO

Improvements

- Formulas to calculate calories, sleep level, calories goal and step goal
- Design
- Change the scene by rotating the activity tracker
- Change time and date
- Add more functionality, like an alarm, etc.
- ...

THANK YOU