Group 2: Savannah Floyd, Patrick Knight, Phillip Wilson

CSCI 360-01

Professor Mountrouidou

1 December, 2017

Health Monitor System Final Report

**Introduction**

Throughout the semester, we have been developing a health monitoring system similar to that of a Fitbit. Systems like the Fitbit aim to give the user of the system a simple and effective way to monitor their everyday habits in a portable and easy to use device. Some of these habits include daily steps taken, calories burned, number of hours slept, etc. It also gives them the ability to check their current heart rate, as well as function as a wrist watch, allowing the user a quick way to check the time and date. The combination of these features is what makes health monitoring systems so great. The portability of them adds to their convenience, and overall will help to improve the daily life of those utilizing them. With our health monitor system, we aimed to replicate this. From the inception of this system, all the way through completion, we aimed to provide all of the basic functionality that users of similar systems desire and utilize most frequently. In this report, we will go through various aspects of the system design that have been developed throughout the course of this semester and discuss why each aspect was chosen and designed the way that it was, as well as what features ultimately were not in our scope due to certain constraints and therefore were not implemented they way they were presented originally, if at all.

**Important things to Note**

While it is evident throughout the entirety of this document that the system would have a sleep monitoring functionality, this specific feature has been put on the backburner and was not fully fledged out in our final design. Time constraints put a large hinderance on our ability to fully implement every feature of the system as desired. With that being said, the inclusion of sleep monitoring in each iteration of our design was crucial for our intended system design, and we stand by its inclusion.

Also, an alarm clock was implemented into our system, which is not reflected in the various diagrams in this report. Also, screens were added into the GUI that allow the user to enter their information before being taken to the clack interface. These were extra additions that were added after the majority of our diagrams were made.

Time restraints and conflicting schedules played a large part in our inability to get the system fully fledged out the way we originally intended it to be.

**Vision**

Our initial vision for this system was a wearable device that would give users a convenient way to track their day-to-day information, such as their current heart rate, total daily steps, total daily calories burned, and their sleeping habits, as well as use as a watch with time and date capabilities. The user would ideally use this information to keep track of their health and make changes in their daily habits, such as increasing their daily steps taken.

Our target user-group is anyone who wishes to use one or more features of the system. There are no target demographics; anyone who is old enough to wear a watch and read the information onscreen will be able to utilize the system. Variables such as gender, race, nationality, education, or knowledge of the system do not apply here.

**Use Cases**

Deciding on our initial use cases was somewhat of a challenge. Going in, we weren’t entirely sure on how broad or specific a use cases had to be.Our original use cases included Tracking an Activity and Monitoring Sleep, and were not fleshed out to their full extent. After some more deliberation and a better understanding of what scope use cases covered, we ended up with a solid group of use cases that cover the majority of the desired functionality that our system should have. These use cases include:

* Tracking an Activity, which includes 3 sub-use cases:
  + Monitor Heart Rate
  + Monitor Calories Burned
  + Monitor Steps Walked
* Display Clock
* Track Sleep Data
* Setup Device

Tracking an activity was first and foremost the most important use case to develop, as it covers the broadest scope of our system’s functionality. We decided that rather than making the use cases for heart rate, calories burned, and steps walked their own separate use cases, it would be a better decision to have them all connected under the umbrella scope of ‘Tracking an Activity.’ Many details in each of these sub-use cases are the same, and these similarities are reflected in the main use case of Tracking an Activity. By allowing the user to keep track of their current heart rate, it would give them an idea of their resting heart rate and would allow them to make changes to their lives accordingly, whether they needed to raise or lower this rate. The same idea can be applied for steps taken and calories burned; based on observed data, the user could decide to increase steps taken and calories burned if they feel that their current daily average was not sufficient enough.

The second most important use case we developed was Display Clock. Since this system will be used as a watch for every single user, whether they utilize any other features or not, it was important to develop a use case that would cover the scope of this functionality.

Tracking sleep data was one of the original use cases that we created, and was the catalyst that cause sleep monitoring to become an integral part of our design. The idea behind monitoring sleep activity is to give users an idea of the amount of sleep they’re getting through the night so that they may view patterns in their sleep cycles and make adjustments accordingly, depending on if they feel they need to get more or less sleep.

The final use case, which was only developed recently, was the Setup Device use case. The first thing a user will do whenever they turn on the device for the first time is set up their user profile, whether that be through the system’s interface or through a separate application. It was easy to forget that this use case existed when originally designing them, as it is something that is only done once in most lifecycles of the system. The only time it would be repeated would be if the system was factory reset by the user and had to be set up again. Below is the Setup Device use case, which shows this new use case in its entirety and also demonstrates the format we used for every other use case we derived:

**Use Case 1:** Track an Activity

**Level:** User Goal

**Primary Actor:** Fitbit User

**Stakeholders and Interests:** Fitbit User - Wants to be able to view their activity information for the current day. Activity information includes, number of steps, distance, minutes active, minutes stationary.  **Preconditions:** The fitbit has been setup for the specific user, and the user is correctly wearing the device.

**Postconditions:** The activity information is displayed to the user on the fitbit.

**Main Success Scenario:**

1)The user puts on his/her fitbit and powers it on.

2)The user performs an activity such as running.

3) During the activity the fitbit monitors and displays heart rate(Use Case 1A), calories burned(Use Case 1B), and steps walked(Use Case 1C).

4) The user after completing the physical activity clicks on the fitbit to view the average heart rate, calories burned and steps walked from the activity.

**Extensions:**

* View Extensions for UC1A, UC1B, UC1C.

**Technology and Data Variations:**

* The Distance displayed in the United States, Liberia, Myanmar should be in Imperial Units. Everywhere else distance should reflected using the Metric system.
* The user may manually enter the activity and duration of the activity if they were not wearing the fitbit (i.e., if the user was on a treadmill running for 30 minutes). This will automatically calculate the amount of calories burned (UC1B).

**Special Requirements:**

* Language internationalization on the text displayed.
* Clear UI that displays the user’s activity information

**Frequency:** Continuous

**Use Case 1A:** Monitor Heart Rate

**Level:** User Sub-Goal

**Primary Actor:** Fitbit Device

**Stakeholders and Interests:** Fitbit Device: Wants to keep track of user’s heart rate during times of activity and no activity. **Preconditions:** The fitbit is set up specifically for the user and is worn during the user’s desired times for tracking heart rate. The clock must be worn so the pulse monitor is against the user’s pulsepoint. The user is wearing the device and has powered it on.

**Postconditions:** The Fitbit UI will display the user’s current heart rate on the display. The fitbit has gathered data on the user’s heart rate.

**Main Success Scenario:**

1) The user continuously wears the clock during the time they want to track their heart rate.

2) The fitbit device is continually taking in heart rate information from the user’s pulse with a heart rate sensor

3) The Fitbit will constantly gather and display the user’s current heart rate, giving immediate feedback to the user.

**Extensions:**

1. At any time, the fitbit shuts down due to power loss.
   1. The user’s data will not be lost due to power loss.
   2. The user must charge the Fitbit to allow the device to power back on.
   3. The user must put the Fitbit back on their wrist with the pulse monitor adjusted to align over their pulse point before the Fitbit will continue monitoring their heartbeat.
2. The Fitbit user is not wearing the Fitbit.
   1. The Fitbit will not track anything (This includes heart rate, steps taken, activities engaged in, etc.) and can only be used for timekeeping purposes (see UC3).
   2. The user must put the Fitbit on before it will continue tracking their heart rate.
3. The Fitbit user is wearing the Fitbit, but does not have the pulse monitor aligned with the pulse point in their wrist.
   1. The Fitbit will not be able to detect their pulse and therefore will not be able to keep track of the user’s constant heart rate.
   2. The user will need to adjust the clock so the pulse monitor is aligned with their pulse point.
   3. The Fitbit will then continue to keep a constant track of the user’s heart rate.
4. The user’s heart rate becomes too high and is now a health risk.
   1. The Fitbit will vibrate to alert the user of a notification.
   2. An on screen prompt will tell the user that their heart rate is too high.
      1. If the user is doing a physical activity, they should slow down to lessen their heart rate before continuing.
      2. If the user is stationary, therefore not doing anything strenuous to cause their heart rate to become too high, they should seek out medical attention immediately.
   3. Once the user’s heart rate has returned to a healthy range, they will be able to continue tracking their heart rate as normal.
5. The user’s heart rate becomes too low and is now a health risk.
   1. If the user is asleep and their heart rate is low because of that, the Fitbit will not notify them.
   2. If the user is awake, the Fitbit will vibrate to alert the user of a notification.
      1. An on screen prompt will tell the user that their heart rate is too low.
         1. If the user is stationary or in a relaxed state and does not feel the need to seek medical attention, then they do not have to.
         2. If the user is stationary, in a relaxed state, or in an active state and feels that it is necessary, they should seek out medical attention.
      2. Once the user’s heart rate has returned to a healthy range, they will be able to continue tracking their heart rate as normal.

**Special Requirements:**

* Clear UI that can be easily read by the user so that they can accurately read their constant heart rate at any given time.
* Backlit screen so the user’s on screen information can be read at any time.
* Sensitive sensors that can accurately monitor the user’s pulse on their wrist.

**Frequency:** Constant

**Use Case 1B:** Monitor Calories Burned

**Level:** User Sub-Goal

**Primary Actor:** Fitbit Device

**Stakeholders and Interests:** Fitbit User, Fitbit Device **Preconditions:** The user has correctly set up for the user with accurate information. The fitbit device is powered on and is being worn by the fitbit user.

**Postconditions:** The fitbit UI will display the calories burned. The fitbit device has gathered data on the calories burned by the user.

**Main Success Scenario:**

1) The user continuously wears the clock during the time the device is tracking the calories burned.

2) The fitbit will calculate the calories burned based off of the user’s activity, the duration of the activity, and the user’s weight.

3) The fitbit will constantly update the calories burned whether the user is walking or doing intense activity.

4) The fitbit will display the calories burned so far for the day to the user on the UI display.

**Extensions:**

1. At any time, the fitbit shuts down due to power loss.
   1. The user’s data will not be lost due to power loss.
   2. The user must charge the Fitbit to allow the device to power back on.
   3. The user must put the Fitbit back on their wrist before the fitbit will begin monitoring Calories burned
2. The user was not wearing the fitbit during an activity.
   1. The user will be able to manually enter the activity and duration of the activity (if they know that information) and the amount of calories burned will automatically be calculated.
3. The fitbit device resets the calories burned data at the default or user specified time every 24 hours.
4. The user updates their weight.
   1. The total amount of calories burned for the day will be recalculated to display a more accurate reading based off of the user’s new weight.
   2. The calories burned will continue updating throughout the day with the new weight included in the calculations, and will continue using this calculation until the user updates their weight again.

**Technology and Data Variations:**

* The user could manually enter calories burned if they were not wearing the fitbit during their activity if they are able to get the information from another source (i.e. a treadmill).

**Special Requirements:**

* The user’s weight must be up-to-date for an accurate calculation.

**Frequency:** Continuous

**Use Case 1C:** Monitor Steps Walked

**Level:** User Sub-Goal

**Primary Actor:** Fitbit Device

**Stakeholders and Interests:** Fitbit User, Fitbit Device **Preconditions:** The user has correctly set up for the user with accurate information.

**Postconditions:** The fitbit UI will display the calories burned. The fitbit device contains data on the steps walked by the user.

**Main Success Scenario:**

1) The user continuously wears the clock during the time the device is tracking the calories steps walked.

2) The fitbit device is continually taking in accelerometer data to calculate the current number of steps walked.

3) The Fitbit will constantly gather and display the user’s steps walked, giving immediate feedback to the user.

**Extensions:**

1. At any time, the fitbit shuts down due to power loss.
   1. The user’s data will not be lost due to power loss.
   2. The user must charge the Fitbit to allow the device to power back on.
   3. The user must put the Fitbit back on their wrist before the Fitbit will continue monitoring their steps walked.
2. The user enters a moving vehicle
   1. The fitbit should recognize that while the user is accelerating steps aren’t being taken, and should not log additional steps
3. The fitbit device resets the calories burned data at the default or user specified time every 24 hours.

**Technology and Data Variations:**

1. User could manually enter their steps taken based on another device (such as their smartphone or steps counted on a treadmill) if they were not wearing their Fitbit.

**Special Requirements:**

**Frequency:** Continuous.

**Use Case 2:** Display Clock

**Level:** User Goal

**Primary Actor:** Fitbit User

**Stakeholders and Interests:**

Fitbit User: Wants to know the current time and date for various reasons

Fitbit Device: wants to know the current time and date for timestamping activities/events **Preconditions:** The user has set up the fitbit and has correctly set the time, date and their timezone.

**Postconditions:** The current time and date are displayed on the fitbit device. Time and date includes the hour, minutes, seconds, day, month, and year.

**Main Success Scenario:**

1. The user has their fitbit powered on.
2. The user navigates to the “Clock” menu.
3. The user can see the current date(day, month, year) and time(hour,minutes,seconds).

**Extensions:**

1. The user has switched time zones.
   1. The user must manually enter in the correct time
   2. Or the user plugs the fitbit into a computer and the correct time is synced
2. The device runs out of battery or is powered off
   1. The device goes into a low power mode where the current time is still being kept
3. The user is not wearing the device
   1. The user can still use the device for timekeeping purposes
4. The user wants to use the clock as a stopwatch
   1. The user clicks the stopwatch button
   2. A stopwatch starting at 00(h):00(m):00(s) is displayed
   3. The clock runs up until the user clicks the stop button or the display reaches 99(h):99(m):99(s)
5. The user wishes to change their clock from standard time to military time, or vice versa.
   1. The user will go into their fitbit’s settings
   2. The user will then select the type of time they want to use.
   3. The watch will automatically convert the time to the appropriate time setting, and will continue operating as normal. The time display will remain under this setting until the user changes it.
6. The user wishes to change their date display information to by day/month/year to month/day/year, or vice versa
   1. The user will go into their fitbit’s settings
   2. The user will then select their date display preferences
   3. The date on the interface will update to the new date display preference and will remain in this format unless the user changes the settings again.

**Technology and Data Variations:**

* The date and time data displayed on the clock screen will differ from timezone to timezone.
* The user can select whether the time is displayed in Standard time or Military time.
* The user can select whether to display the date in day/month/year format (as is used in the USA) or month/day/year format (as is used in the UK)

**Special Requirements:**

* The data displayed is updated real time, and is displayed in a readable manner for the user
* A backlit display so the user can read the time in any lighting, from being in the sun to being in total darkness.

**Frequency:** Continuous

**Use Case 3:** Track Sleep Data

**Level:** User Goal

**Primary Actor:** Fitbit User

**Stakeholders and Interests:**

Fitbit User: Wants to be able to keep a recorded log of their sleeping habits, including their time asleep and their time spent during each stage of the sleep cycle.

Fitbit (Company): Wants users to track their sleeping habits so they can collect valuable data on their user’s activities. **Preconditions:** The Fitbit has been set up specifically for the user and is worn while the user is sleeping.

**Postconditions:** The Fitbit will return a log of the user’s sleep from the time they were asleep.

**Main Success Scenario:**

1) The user puts on his/her Fitbit and powers it on.

2) The user falls asleep while wearing the Fitbit and is asleep for a minimum of one (1) hour.

3) During sleep, the Fitbit will record the user’s specific sleep cycle and duration of sleep. This includes time spent asleep, time spent in the REM stage of sleep, time spent in the other stages of sleep,

4) The user will be able to view their personal sleep data after they wake up. The user will be able to keep track of their overall sleeping habits over various days, and will be able to view trends in their sleep cycles compared to other Fitbit users.

**Extensions:**

1. At any time, the Fitbit shuts down due to power loss.
   1. The user’s sleep data already recorded will not be lost, and is recoverable.
   2. The Fitbit will not continue to track sleep until the Fitbit is charged and attached to the user’s wrist again.
2. The user falls asleep using their fitbit, but is only asleep for less than one (1) hour.
   1. The fitbit will not track this sleep data.
   2. The user must fall asleep and stay asleep for a minimum of one (1) hour for the fitbit to begin tracking their sleep data.
   3. Colors on the UI that are not too harsh on the eyes, as many users who actively track their sleeping habits may view this information right after waking.
3. The user is not wearing their fitbit while sleeping, but still wants the sleep information to be logged onto their device
   1. The user can manually enter the hours they were asleep
   2. This will not be able to tell them useful information about their sleep cycle (i.e. REM sleep and time awake), but will still be able to show them trends in the total amount of time they are sleeping on average per night.

**Technology and Data Variations:**

* The user may manually enter their hours of sleep if they were not wearing the fitbit while they were sleeping. This will clearly not allow the user to view all of the details of their sleep (time in the REM cycle, time awake, etc.) but could still be useful in showing them their total hours of sleep per week. This would also allow the user to enter times asleep that lasted for less than one hour.

**Special Requirements:**

* Clear UI that displays the user’s sleep information
* Colors on the UI that contrast well together so the user is able to easily view and identify their personalized charts that display their sleeping habits
* Backlit screen so the user can read the information in the dark.

**Frequency:** Every time the user wears their Fitbit while sleeping for a minimum of one (1) hour; Daily.

**Use Case 4:** Setup Device

**Level:** User Goal

**Primary Actor:** Fitbit User

**Stakeholders and Interests:**

Fitbit User: Wants to be able to set up the device with their personal specifications (weight, age, gender, etc.)

Fitbit (Company): Wants to collect user data for device statistics. **Preconditions:** The Fitbit must be powered on. It can or cannot be fastened on the user’s wrist during setup.

**Postconditions:** The Fitbit will take the user to the home screen upon completion.

**Main Success Scenario:**

1. The user powers on their fitbit device.
2. The fitbit will greet the user, then prompt them to enter their personal information that is required to give a more accurate calculation of calories burned.
   1. The user will enter their age.
   2. The user will enter their sex.
   3. The user will enter their weight.
   4. The user will enter their height.
3. Once this setup has been completed, the user will be taken to the home screen (in this case, the clock interface) and they may begin using their device.

**Extensions:**

1. If the user’s device dies during setup
   1. The information entered so far will not be saved
   2. The user must charge the device and power it back on
   3. Once the device is powered on again, the user may restart the setup process
   4. The user’s information will not be saved in the memory of the device until they completely finish the setup process
2. If the user enters information that is not appropriate for the device (i.e. entering a weight that is too low, like 0 lbs)
   1. The device will not accept this information and will not allow the user to continue with the setup process until they have entered something different

**Technology and Data Variations:**

* Depending on what measurement system the user uses, whether it be based on personal preference or their country of origin, certain variables may be displayed differently (i.e. pounds or kilograms)

**Special Requirements:**

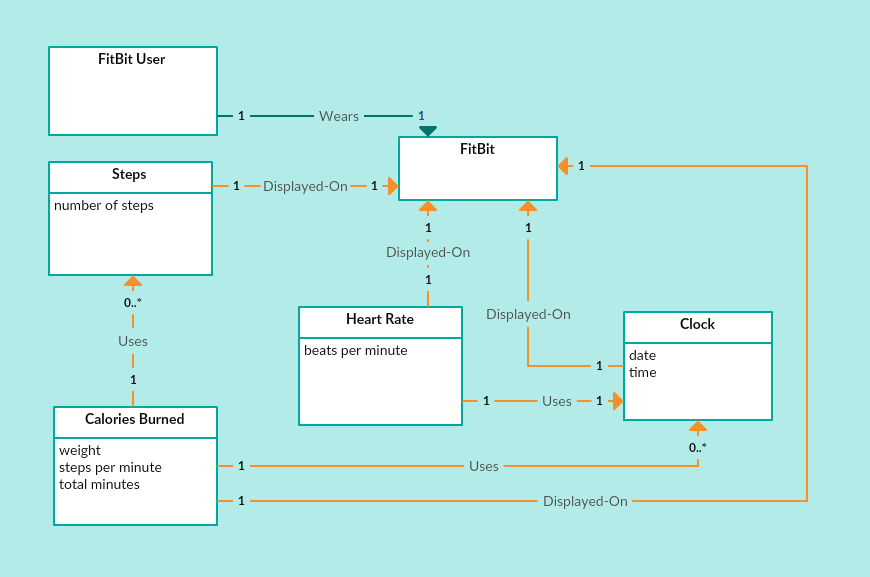
* Display must be large enough that the user may have room to enter this information with little or no room for error

**Frequency:** Once upon initial startup of the device. Will have to be repeated if the device is ever factory reset.

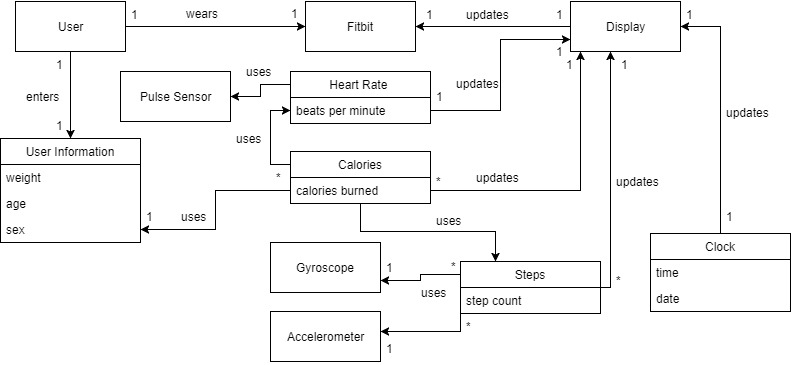
We believe that our use cases cover the scope of our system’s intended functionality. These features are the ones that are used most often by users of the system, and are therefore first and foremost what features need to be included in the system above all else. While not every feature detailed in these use cases was implemented, the idea behind their inclusion remains the same.

**Domain Model**

When designing our domain model, we ran into a similar problem as with designing our use cases; how broad or narrow it had to be. We also had little knowledge on how to implement low-coupling in the design, which added to our overall struggles with this portion of the design process. Our original domain model, which is pictured below, was tightly-coupled and did not include all of the sensors (gyroscope, pedometer, heart rate sensor) that should have been there. We assumed that everything needed to be coupled to the Fitbit class since that was the system we were designing for.



After reevaluating our design and adding in the sensors, our new domain model, pictured below, shows a much better understanding of implementing design patterns as well as a greater knowledge of how to design an appropriate domain model. There is still high coupling, as is apparent on the Display class, but we felt that for the purpose of the domain model, this high coupling was necessary. The addition of the pulse sensor, gyroscope, and accelerometer give a better understanding of how each part of the system will interact.



Although the original domain model design was not entirely reflective on how our overall system would be designed, it became an integral part of how we would later develop other aspects of our system’s design. Understanding how each part of the physical system would interact was crucial in giving us the ability to appropriately design the code that would simulate these interactions.

**Operation Contracts**

Our original operation contracts were designed with a much simpler functionality in mind. We created them with a simplistic view of the system in our minds. Originally when we made these, they were more focused on the physical state of the user and system, rather than the more technical aspects of it. The original operation contracts can be found below:

OC1:

Operation: getTime()

Cross References: Use Case: Display Clock

Preconditions:

* The fitbit clock object has been correctly initialized with time, date and time zone fields

Postconditions:

* An instance of time was created
* The timeIsDisplayed attribute of the clock was set to true

OC2:

Operation: getDate()

Cross References: Use Case: Display Clock

Preconditions:

* The fitbit clock object has been correctly initialized with time, date and time zone fields

Postconditions:

* An instance of date was created
* The dateIsDisplayed attribute of the clock was set to true

OC3:

Operation: getHeartRate(pulse)

Cross References: Use Case: Monitoring Heart Rate

Preconditions:

* The user object has been initialized with the correct weight, height, and age fields.
* The fitbit device is aligned correctly on the user.

Postconditions:

* A pulse attribute p was created
* p was associated with getHeartBeat
* p was associated with heart beat

OC4:

Operation: getSleepData(heartRate, weight, height)

Cross References: Use Case: Track User’s Sleep

Preconditions:

* The user object has been initialized with the correct weight and height fields
* User is wearing their fitbit correctly on their wrist

Postconditions:

* An instance of sleep sl was created
* sl was associated with sleepData
* The sleepData attribute of the fitbit system was updated with either REM or Deep sleep data

OC5:

Operation isAsleep(heartRate)

Cross References: Use Case: Track User’s Sleep

Preconditions:

* The user object has been initialized with the correct weight, height, and age fields.
* The fitbit device is aligned correctly on the user.

Postconditions:

* An instance of getHeartRate hr was created
* hr was associated with isAsleep
* A boolean was returned to the system based on the value of hr

OC6:

Operation: displaySleepData(sleepData)

Cross References: Use Case: Tracking User’s Sleep

Preconditions:

* The sleepData attribute contains data
* The isAsleep method returns false

Postconditions:

* An instance of sleep sl was created
* sl was associated with sleepData
* The sleepDataIsDisplayed attribute of the fitbit was set to true

OC7:

Operation: trackCaloriesBurned(weight, pulse, steps)

Cross References: Use Case: Tracking Calories Burned

Preconditions:

* The user object has been initialized with the correct weight, height, and age fields.
* The fitbit device is aligned correctly on the user.

Postconditions:

* The currentCaloriesBurned attribute was displayed to the user
* The currentCaloriesBurned was continuously updated throughout the 24 hour period
* After 24 hours the currentCaloriesBurned was reset to 0

OC8:

Operation: initialSetup()

Cross References: Use Case: Setup Device

Preconditions:

* The fitbit device has been powered on

Postconditions:

* An instance of user u was created
* An instance of weight w is created
* U is associated with w
* An instance of age a is created
* U is associated with a
* An instance of sex s is created
* U is associated with s
* An instance of height h is created
* U is associated with h
* The information associated with u is confirmed by the user and the system is updated

The operation contract created most recently was that of the Setup Device one. It was made as an addition to go along with our Setup Device use case. It is the one that already most closely matches our current system, but it still requires an update.

New operation contracts were created to match the current implementation of our system. They exhibit a better understanding of how the functions within our system interact. The most important of these can be viewed below:

OC1:

Operation: createUser(float weight, float height, char sex)

Cross References: UC4: Setup Device

Preconditions:

* The health monitor device has been correctly powered on and initialized

Postconditions:

* An instance of user u was created
* An instance of weight w was created
* An instance of height h was created
* An instance of sex s was created
* u was associated with w
* u was associated with h
* u was associated with s
* The information associated with u is confirmed by the user and the system is updated

OC2:

Operation: startMonitoringPulse()

Cross References: UC1A: Monitor Heart Rate

Preconditions:

* The health monitor device has been correctly initialized with the correct weight, height, and sex fields
* The health monitor device is aligned correctly on the user

Postconditions:

* An instance of currentHeartRate chr is created
* chr is associated with the HeartRateMonitor
* chr information is sent to the Display

OC3:

Operation: startMonitoringSteps()

Cross References: UC1C: Monitor Steps Walked

Preconditions:

* The health monitor system is aligned correctly on the user

Postconditions:

* An instance of currentSteps cs is created
* cs is associated with the StepsMonitor
* cs information is sent to the Display

OC4:

Operation: monitorCalories()

Cross References: UC1B: Monitor Calories Burned

Preconditions:

* The health monitor device has been correctly initialized with the correct weight, height, and sex fields
* The health monitor system is aligned correctly on the user

Postconditions:

* An instance of caloriesBurned cb is created
* cb is associated with the CaloriesMonitor
* cb information is sent to the Display

OC5:

Operation: startMonitors()

Cross References: UC1: Track an Activity

Preconditions:

* The health monitor device has been correctly initialized with the correct weight, height, and sex fields
* The health monitor system is aligned correctly on the user

Postconditions:

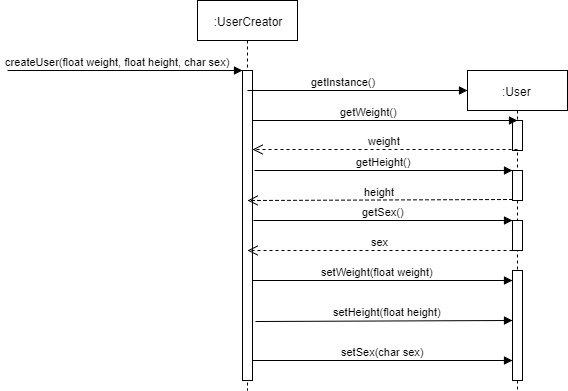
* An instance of HeartRateMonitor hrm was created
* An instance of StepsMonitor sm was created
* An instance of CaloriesMonitor cm was created
* hrm was associated with startMonitors()
* sm was associated with startMonitors()
* cm was associated with startMonitors()
* The information obtained in startMonitors() was transferred to the Display

**Sequence Diagrams**

When creating our sequence diagrams, we used our class diagram as a guide in how different functions should interact between classes. We ended up having to make a lot of changes to our SD’s over various deliverables, but overall, creating them gave us a better understanding of how our classes should interact and how various design patterns worked.

Below are our new sequence diagrams that were created to match the current implementation of our system:

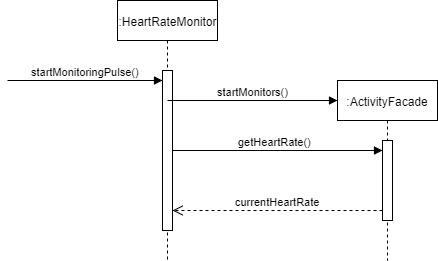
User Creation Sequence Diagram:

****

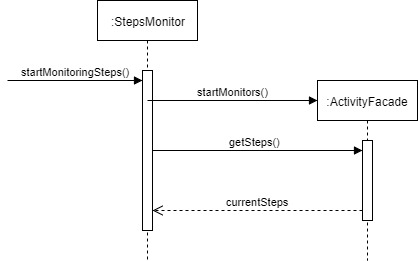
This SD shows the interaction between the UserCreator and User classes. The user creator class calls for a new user to be made. Once this happens, an instance of user is called using the getInstance() function. The user at this point would be able to enter their weight, height, and sex into the system using the various set functions. These variables can be viewed at any time using the various get methods. The various get functions would also further be used by the ActivityFacade class to help calculate calories burned. The UserCreator class could be seen as having implemented the Singleton design pattern, as it allows for one user with one set field of variables to be instantiated, and for that information to be used within other classes. Ideally, no other user should be able to be created within our system since in real life only one person would be using a Fitbit at a time.

The User class could also be seen as an Information Expert, since it holds all of the information about a user that the system would need to know to use in any other class within the system.

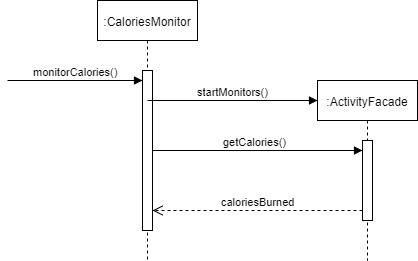
Monitor Heart Rate Diagram:

****

Monitor Steps Sequence Diagram:

****

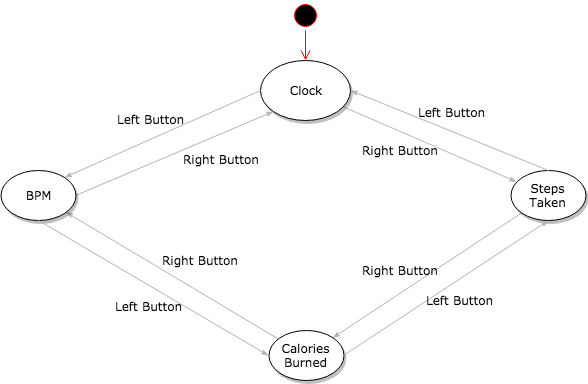
Monitor Calories Burned Sequence Diagram:

****

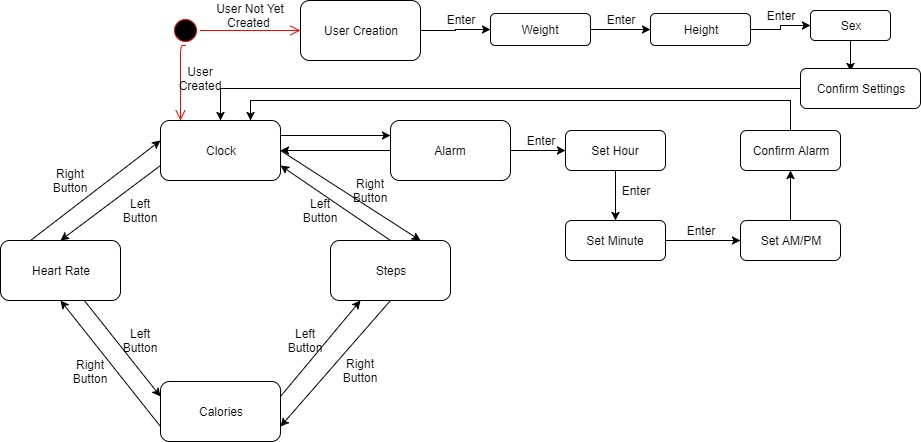
The Monitor Heart Rate, Monitor Steps, and Monitor Calories Burned sequence diagrams are all essentially doing the same thing. There is a call by the respective monitor to start monitoring the specific variable. Then, the ActivityFacade class calls the startMonitors() function, which starts up each of the monitors and begins collecting data. The getHeartRate(), getSteps(), and getCalories() functions all, respectively, do exactly as they sound; they get the current number of each variable and returns that number. The use of the Facade design pattern helps in implementing these Monitor classes. Each class is very similar in execution, and would be fairly easy to confuse, so by using the ActivityFacade class as a bridge between the three and the display, we are provided with an easy way to keep up with all of the incoming information, as well as keep it all in one place to avoid confusion.

**State Diagram**

Our first state diagram shows the basic functionality of our system, and how each part of our UI works together to show the implementation of the system. Once the system is turned on, the first screen visible to the user is the clock interface. There are left and right buttons that allow the user to cycle through the screens and view their current heart rate, their daily calories burned, and their daily steps taken.

****

Since finalizing the implementation of our system, we have designed a new state diagram, which can be seen below:

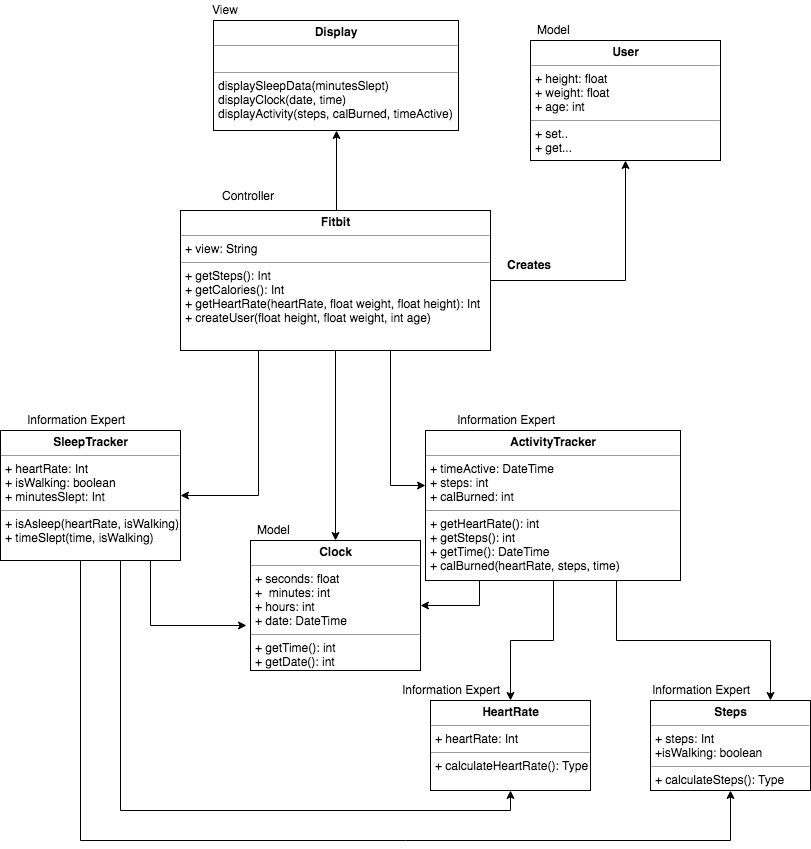


This SD has all of the same functionality of the previous one, with User Creation and Alarm setting added into the equation.

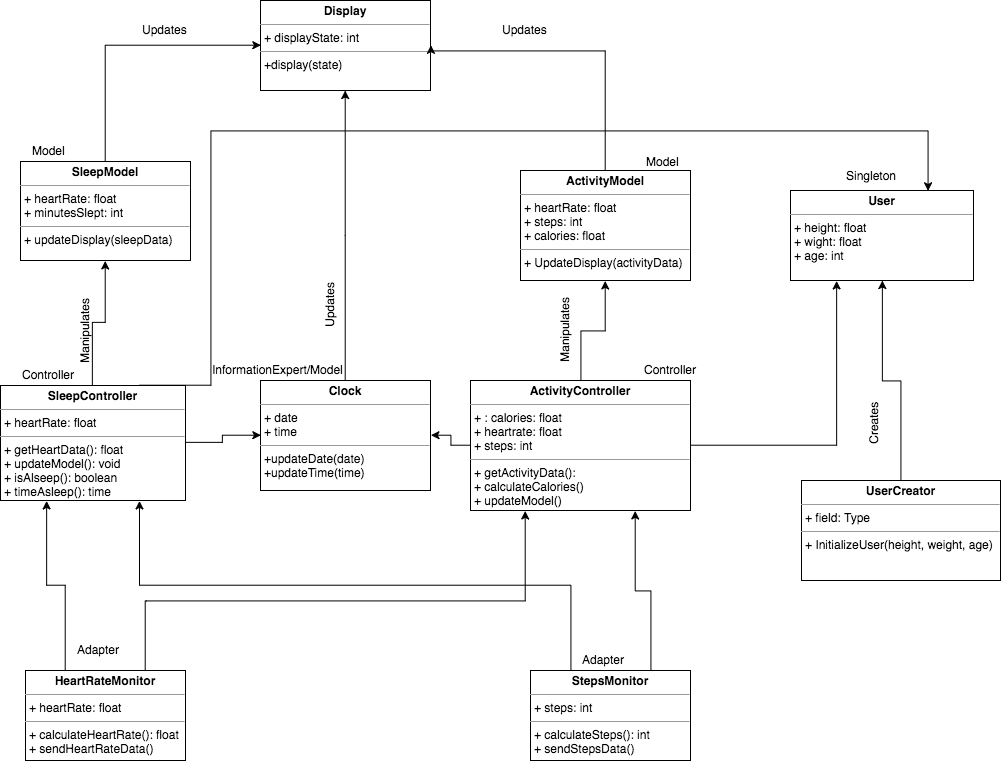
For our system, we wanted to implement a minimalist design that still included all of the basic functionality of a standard health monitoring system. We have definitely accomplished that with the design of this system.

**Class Diagram**

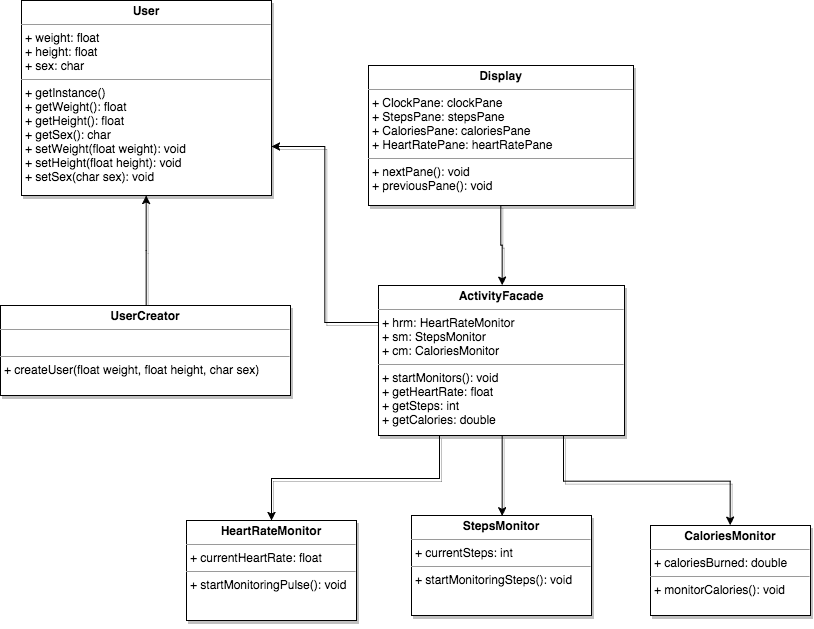
Our original class diagram was designed much in the same way as the domain model in that it was too tightly coupled between various classes. While we did design this class diagram with various design principles in mind (model, controller, information expert, etc.), it does not show that we had a good understanding of how to implement these design principles, and overall was a very flawed design. This original diagram can be seen below:



After reevaluating our original class diagram, we created one that was more in tune with what our system would likely do and how it would ideally function. It was much less coupled and included controller classes, which helped to control what information was getting sent where within the system and helped to support various design principles. This class diagram can be seen below:



This class diagram did a good job at portraying the various classes of our system and what each one was responsible for doing through the use of Gang of Four design patterns. However, due to the drastic change in our code once we began implementation, this class diagram is no longer an accurate representation of our design. A new class diagram, pictured below, was created to better represent our finalized coded implementation of the system:



The UserCreator class takes in information about the user to create a user profile. It can be seen as having used the Singleton design pattern, as it allows for the creation of one user whose information will then be used in other parts of the system.

The User class is comprised of getters and setters that allow for the retrieval and manipulation of user data. These data elements include the weight, height, and sex of the user. This information in theory would be used to help calculate calories burned. This class could be seen as an information expert, since it holds all of the necessary information on a user that would be used throughout the system.

The Display class is responsible for creating the user interface using JavaSwing. It creates different panes for each of the screens the user can cycle through (the time and date screen, heart rate, steps taken, and calories burned, and has two functions for cycling through these pains; nextPane() and previousPane(). It takes in information provided by the ActivityFacade class to display to the user.

The HeartRateMonitor class is responsible for taking in heart rate information via an artificial pulse that has been implemented into the system. This was done through the use of reading a text file with various beats per minute (BPM) and selecting one to display.

The StepsMonitor class is similar to the HeartRateMonitor class in that it uses information read from a text file to display the user’s step information.

The CaloriesMonitor class monitors the calories burned by the user based on a combination of factors, including steps taken, time elapsed, and weight of the user.

The ActivityFacade class is responsible for starting the monitors, collecting the data provided by those monitors, and supplying that information to the Display class in order for the information to be visible in the user interface. We decided to use a facade pattern to implement this because we wanted the code to be organized in a way that it would be simple to obtain the information from various classes in order to update the GUI. It has given us a simplified interface to access the three Monitor classes and transfer the information gathered from those classes. It has also helped in making the code and output more readable, since all of the information is being gathered into one place.

This class diagram shows a much simpler and functional interface for our system. Our previous CD was full of controller and model classes that ended up being unnecessary in our final implementation. Perhaps a larger system would have been able to make use of this number and types of classes, but our system did not.

**Reflection**

Looking back, there are many aspects of designing and implementing this system that could have been done better or in a different way. For example, while we had the best intentions in including sleep monitoring as a function of our system, we were at a loss for how to implement it to the best of our abilities within the time given. One observation that could be made is that it would be helpful if code components of the project would begin earlier in the semester, rather than trying to fit it all into the last two weeks or so. It would be much easier and less stressful this way. Another observation is that Java Swing was not an entirely intuitive coding API to use for this project. By having the ability to choose our own preferred API when completing this project, it could have been a much smoother process. Perhaps the use of JavaFX would have been better.

**Conclusion**

Overall, the knowledge and skills obtained throughout this semester taking this course has been valuable. Learning the development process and the steps needed to implement this system was challenging, but has ultimately been a rewarding experience in expanding our knowledge and giving us skills to use in the future. Ultimately, we completed our goal of implementing a replica of a health monitoring system. While some features were excluded, ultimately, the system does the majority of what we wanted and is a good representation of the effort we have put into this project throughout the semester.