Implementation and Report of EdinBike Bike – Hire Scheme

Nicholas Georgiou – s1342226
John Baris - s1443483

EdinBike Project Report

The following report contains an overview of the system and what is needed from it. It then goes on to describe the design and tests of the system and why we have added certain methods and classes.

EdinBike Overview

The system will consist of several bike docking stations around the city where a user can register and
then hire a bike for which they are charged. Initial charge is £1 for the first 30 minutes and then £2 for each additional 30 minutes. Once they are done they can return the bike to any docking station in the city with available docking points. If no docking points are available at the station the user returns the bike, they report this and the system produces a map with the closest alternative docking stations, extending their time for 15 minutes with no additional charge. If a bike is faulty the
user can report this by returning the bike and pressing the fault button. The overall system is supervised from an operations HUB. Here all the information regarding the docking stations is displayed and problems that occur are dealt with, by the personnel. Reports can also be issued at the Hub containing information about all the docking stations.

Purpose of EdinBike

A number of cities around the world have invested in systems which promote using a bicycle to move around the city. As Edinburgh is one of the major cities in Europe and attracts many tourists it has been decided that a bike renting system will be implemented here as well. The main aim is to provide users an easy to understand system for renting out bicycles.

EdinBike Event List

Event Name	Input and Output	Summary
1.User presses registration button.	Personal Details (IN)	Store user's personal details as entered by a new user.
2. User enters credit card into card reader.	Credit Card in card reader(IN)	System verifies that the user's card is valid
3.Key with unique ID issued	Key issued (OUT)	Once user registers, unique Key is issued to user
4. User hires a bike	Key reader triggered with Key(IN) Bike Unlocked(OUT)	User places their key on the key reader on Docking Point. Bike unlocked
5. User returns bikes	Bike sensor on DP triggered(IN) Green Light flashes (OUT)	User inserts bike into DP, bike ID reader triggered. Notes that bike returned. Green Light flashes
6. User returns faulty bike	Bike sensor on DP triggered(IN) Red Light flashes (OUT)	User inserts bike into DP, bike ID reader triggered. User presses fault button Red light flashes. Bike noted as faulty
7. User returns to full docking station.	Key reader on terminal activated(IN) Map displayed on terminal(Out)	Key reader on terminal activated. Tells system DPs full. Produce map for user with closest available DS
8. User requests a usage report	Usage button pressed and key placed on key reader terminal (IN) (IN) Usage report displayed(OUT)	Usage report button pressed User identified by using unique key. Usage report for user displayed on screen
9. Operator adds a new station	Operator adds DS information(IN) Displays all DS in the system(OUT)	Operator adds all the information regarding the new station and then the display at the HUB shows that the new station added as one more station displayed
10. Personnel request report	Report requested (IN) Report generated (OUT)	Personnel at the HUB request a report about the system. Report is generated and displayed
11. Personnel removes bike	Key reader triggered with Key(IN) Bike Unlocked(OUT)	Key reader detects if the key belongs to staff or a user. Detects user so does not charge
12. Bank Charges	System timer detects it is midnight	Every day at midnight, accounts of users are charged.
13. Wall Display Hub (OUT)	Information displayed on wall at the docking stations occupancy is	Information regarding displayed on a wall

Classes which were implemented

The following information describes the classes in the project which we have implemented as to make the program running. We also note as to which use cases they are related.

Hub

The Hub is the central part of this system. Almost all tasks and methods will pass through the Hub as it is the 'heart' of the system.
The Hub is where all the docking station, user and bike details are stored. It contains methods for adding a new user to the system, adding a new docking station, adding a new bike into the system, viewing the statistics of the system, retrieving specific docking station information and getting information about a specific user. The addUser method is implemented when a user starts the registration process (RegisterUser use case). AddBike method is used when a new bike is being added to the system, and the addDStation is used when a new DStation is being added to the System. This covers the AddDstation use case. There are also to methods setDistributor and setCollector, where setDistributor collects the input and the setCollector displays the output. The method processTimedNotification is implemented so that we can determine whether a docking point has a high or a low occupancy. This method contributes to the viewOccupancy use case.

Hub Display

The Hub display is where all the information a Hub operator or a member of staff may want to know is displayed. The hub display holds a method showOccupancy which displays the percentage occupancy of the various docking station, information essential to not having a docking station with little or no bikes and another with full docking points. This method covers the ViewOccupancy use case. There is also a method noDStations which displays all the docking stations the system contains, which is updated when a new station is added. This is one way in which an operator knows that a new docking station has been added as the number of docking stations will be one more than was before. This is part of the AddDStation use case. The showStatus method is implemented to display the status of all the docking stations which are part of the system, and also the status of the bikes which are present in the docking station. This is crucial in knowing which bikes are faulty and so will need repairing. This information is displayed in the report the operator asks for. This is part of the ViewStatus use case.

Hub Terminal

The hub terminal class is used to call the methods which are in the Hub Display as the hub terminal is where the personnel will add or request information to and from the system. There is a method addStation which will be used to add a new station into they system when needed.

Docking Station

This class contains all the information needed for a docking station. It holds the docking station's location, how many docking points it has and instances of the KeyReader which reads the user's key when it is inserted, the cardReader which reads the user's card, the KeyIssuer which dispenses the key and a touchScreen where information is displayed.
This class also includes the method hireABike which increases the number of available docking points at that docking station by one and checks which user has hired the bike. There is also the method dockABike which decreases the number of available docking points at the specific docking station and notes who has returned the bike (whether it is a user or staff). The method removeABike is implemented when a member of staff removes the bike for maintenance or to be moved. A method viewUsage (for use case ViewUserActivity), where this method is used to find the usage for each user, which will then be used to display the specific user's usage for that day. There is also the findPoints method (FindFreePoints use case). This method is used for when a user returns a bike to a full docking station and needs to find the next closest docking station with available docking points and also give the user an extra 15 minutes. There is the startRegReceived (RegisterUser use

case) method in this class which is used to store the information which the new user registering inputs. The method adds the details to the corresponding variables in the user and then the new user is stored in the Hub.

DSTouchScreen

This class contains the methods needed to retrieve information which are needed to be displayed onto the touch screen, such as personal details of the new user, display user activity, display closest docking points and a method to display the prompt which encourages the user to choose what option they would like. The showUserActivity method is used to retrieve and display the user who is requesting it's information concerning the usage of bikes for the current day. This method is linked to ViewUserActivity use case. The showPrompt message is implemented to give the user a message on what is needed of them when they are inputing their personal information or requesting a usage report. Therefore it is linked with various use cases. The findDPoints method is connected to the FindFreePoints use case as it needs to display all the closest docking stations with available docking points for the user to return the bike they rented out.

Dpoint

Docking point contains methods in it for when a bike is hired and returned, and if it is faulty or not. The method keyInserted is implemented so that when a valid Key is entered into a key reader, the green okay light will flash, unlocking the bike which is associated with that key reader. The method bikeReturned is implemented to update the system when a bike which was hired is returned. Once the bike is returned its status is changed to docked. This method is linked with the ReturnBike class. The method hireBike does the complete opposite to bikeReturned. Once a user uses a valid key on a key reader, a bike is unlocked and a green light flashes (keyInserted method). The field in the user class Bike is also updated with the bikeID of the bike hired out by that specific user. This method is linked to the HireBike use case. The method fbuttonPressed is linked with the ReportFault use case. This method is for when a user returns a bike to the docking point and it is faulty. This method sets the specific bike's status to faulty which is needed so that staff know to repair it.The removeBike method in this class is linked to the RemoveBike use case where a member of staff will remove a bike from a certain docking station for repair or any other reason. This method detects that the keyID belongs to a staff member and so does not charge the person who has that key.

BikeLock

This class contains two methods, lock and unlock. They are linked with HireBike, ReturnBike and RemoveBike as in all these use cases we need to either lock or unlock a bike. The lock method is for locking a bike and the unlock is for unlocking a bike.

OKLight

The OKLight is linked to HireBike and ReturnBike use cases as the OKLight flashes once a bike is unlocked when a key is placed in a key reader for hiring a bike and for when a bike is returned and locked successfully. This class contains just one method flash() which makes the green light flash when necessary.

RedLight

The RedLight is linked to the reportFault use case as the RedLight flashes once a bike is locked and the fault button is pressed. This class contains just one method flash() which makes the red light flash when necessary.

KeyReader

In this class we receive three different inputs, and have corresponding output. One key input is for when the user wants to hire a bike, another for when they want a usage report and another is for the staff when they are moving bikes.

User
This class contains all the attributes which make up a user, such as their name, keyId, email, bankCard, usage details, date they hired a bike and date returned, the bikeId for the bike they have hired, total usage and which station they hired the bike and to which they returned the station. It mainly contains getters and setters, apart from the method where the user's charges are calculated so that they can be charged. There is also the calcUsage method which calculates the user's total usage for the day which is then used to calculate the charges. These two methods are used for the ChargeUser use case.

Bike
This class contains two attributes for the bikes in the system, its bikeID and the status of the bike, so whether it is faulty, hired or docked.

BankServer
The bankServer class is needed for charging the user's account at midnight each day. It is an output event which displays that the payment has gone through. The method for charging the user is in the Hub and done automatically at midnight. The bank covers quite a bit of the process for charging the actual bank accounts.

AddDStationObserver
This class contains a method addStation which adds a new station to the list with Docking stations in the Hub and the displayDStation method is the input at the Hub terminal to display all the stations currently in the Hub system. The viewStats method displays all the information on the status of the stations in the Hub.

BikeDockingObserver
This class contains the method bikeDocked which is related to the docking of a bike at the docking point. It is linked with the returnBike use case. The method fbuttonPressed is related to the docking of a faulty bike. Linked to the ReportFault use case.

KeyInsertionObserver
This class contains the method keyInserted which is linked to hiring a bike, the removeBike method is linked to staff member removing a bike and the findPoints method is linked to when a user returns to a full docking station and wants to find the nearest docking station.

FaultButton
In this class we implement the fault button on the docking point for when a user returns a faulty bike and presses the fault button. This is linked with the ReportFault use case.

Functional Requirements

Hub
1. Must be able to print out a report with statistics regarding the docking stations on demand
2. Must be constantly running and receiving updates from the docking stations
3. The Hub must be able to keep track of time accurately
4. Must be able to calculate % occupancy for the display to display
5. It must have a user database where all the registered users information will be stored

Docking Station
1. Must be able to distinguish between a key placed on a key reader at a docking point or docking terminal

2. If the key reader is used on the terminal it must be able to display a map of the closest docking stations on the display.
3. When a bike is hired or returned it should update the information held in the system about the number of bikes and available docking stations at that station
4. Must issue and electronic key when the user's information is correct and credit card is valid.
5. Must distinguish between a user's keyID and staff's keyID.
6. Docking points must distinguish between bikes belonging to the system and those not, through the ID reader.

System:
1. Must charge the users correctly.
    1a. £1 for anything less than or equal to 30 minutes
    1 b. £2 for every extra 30 minutes
2. The system should be constantly active whatever time of the day. Must be always available for a user to use
3. The charges made must be accurate to avoid over or under charging some customers

System Tests
In this section we have copied the code for the system tests and placed them here so that we can explain what each one does.

setUpDemoSystemConfig()
Code:
```
public void setupDemoSystemConfig() {
      input("1 07:00, HubTerminal, ht, addDStation, A, 500, 350, 15");
      input("1 07:00, HubTerminal, ht, addDStation, B, 400, 300, 13");
   }
```
This test sets two stations with the following details: name of station, east position, north position and number of docking point.

setupBikes()
Code:
```
public void setupBikes(){
     input ("2 13:00, BikeSensor, B.2.bs, dockBike, bike4");
     expect("2 13:00, BikeLock, B.2.bl, locked");
     expect("2 13:00, OKLight,  B.2.ok, flashed");
     input ("2 13:00, BikeSensor, A.4.bs, dockBike, bike5");
     expect("2 13:00, BikeLock, A.4.bl, locked");
     expect("2 13:00, OKLight,  A.4.ok, flashed");
     input ("2 13:00, BikeSensor, A.7.bs, dockBike, bike6");
     expect("2 13:00, BikeLock, A.7.bl, locked");
     expect("2 13:00, OKLight,  A.7.ok, flashed");
     input ("2 13:00, BikeSensor, B.5.bs, dockBike, bike7");
     expect("2 13:00, BikeLock, B.5.bl, locked");
     expect("2 13:00, OKLight,  B.5.ok, flashed");
     input ("2 13:00, BikeSensor, A.2.bs, dockBike, bike8");
     expect("2 13:00, BikeLock, A.2.bl, locked");
     expect("2 13:00, OKLight,  A.2.ok, flashed");
     input ("2 13:00, BikeSensor, B.2.bs, dockBike, bike9");
     expect("2 13:00, BikeLock, B.2.bl, locked");
     expect("2 13:00, OKLight,  B.2.ok, flashed");
     input ("2 13:00, BikeSensor, B.1.bs, dockBike, bike10");
```

```
      expect("2 13:00, BikeLock, B.1.bl, locked");
      expect("2 13:00, OKLight,  B.1.ok, flashed");
   }
```
This code resembles a member of staff adding a number of bikes with ID's bike4,bike5.. to the docking station,

setupFaultyBikes()
```
public void setupFaultyBikes(){
      input("2 13:15, FaultButton, A.1.fb, pressFButton, bike2");
      expect("2 13:15, BikeLock, A.1.bl, locked");
      input("2 13:17, FaultButton, A.1.fb, pressingButton");
      expect("2 13:17, RedLight, A.1.rl, flashed");
   }
```
This code makes bike2 initial status of faulty.

setupUsers()
```
public void setupUsers(){

       input ("2 10:20, KeyReader, A.2.kr, insertKey, A.ki-1, bike3");
      expect("2 10:20, OKLight,  A.2.ok, flashed");
      expect("2 10:20, BikeLock, A.2.bl, unlocked");
      input ("2 13:00, BikeSensor, B.8.bs, dockBike, bike3");
      expect("2 13:00, BikeLock, B.8.bl, locked");
      expect("2 13:00, OKLight,  B.8.ok, flashed");
      input ("2 19:00, DSTouchScreen, A.ts, viewActivity");
      expect ("2 19:00, DSTouchScreen, A.ts, viewPrompt, insertKey");
       input ("2 19:00, KeyReader, A.kt, keyInsertion, A.ki-1");
      expect ("2 19:00, DSTouchScreen, A.ts, viewUserActivity, HireTime, HireDS, ReturnDS,
Duration (min), 10:20:00, A, B, 160");
   }
```
This code adds a number of users and their information such as usage to the database of users in the Hub.

setupSchedule()
```
public void setUpSchedule(){
       setupBikes();
       expect("8 00:00, HubDisplay, hd, viewOccupancy, unordered-tuples, 6,"
           + "DSName, East, North, Status, #Occupied, #DPoints,"
           + " A,  500,   350,    LOW,        3,       15,"
           + " B,  400,   300,    LOW,        4,       13");
   }
```
This code displays the information regarding the corresponding stations.

System Tests for Target Use Cases

addStation()
```
   public void addStation(){
      logger.info("Starting test: addStation");
      setupDemoSystemConfig();
      input ("2 07:00, HubTerminal, ht, addDStation, C, 450, 280, 25");
      input ("2 07:00, HubTerminal, ht, addDStation, D, 200, 400, 20");
      input ("2 07:00, HubTerminal, ht, displayDStation");
      expect ("2 07:00, HubDisplay, hd, numberOfDS, D, 200, 400, 20, A, 500, 350, 15, B, 400, 300,
```

```
13, C, 450, 280, 25");
      setupBikes();
   }
```
This code is the use case for AddDStation. The Hub operators inputs the new station's location, name and number of docking points on the Hub terminal which is an input device. As a result list of the current docking stations is displayed.

dockBike()
```
public void dockBike(){
      logger.info("Starting test: dockBike");

      setupDemoSystemConfig();
      input ("2 13:00, BikeSensor, B.4.bs, dockBike, bike2");
      expect("2 13:00, BikeLock, B.4.bl, locked");
      expect("2 13:00, OKLight,  B.4.ok, flashed"); }
```
This code is for the use case ReturnBike and AddBike. It takes as input the bikeID which is read by the BikeSensor(input device) and as a result the the bike is locked(output device) and OK light flashes(output device).

RegisterUser()
```
public void registerUser() {
       logger.info("Starting test: registerUser");

       setupDemoSystemConfig();

       // Set up input and expected output.
       // Interleave input and expected output events so that sequence
       // matches that when describing the use case main success scenario.
       logger.info("registerUser");

       input ("2 08:00, DSTouchScreen, A.ts, startReg, Alice");
       expect("2 08:00, CardReader, A.cr, enterCardAndPin");
       input ("2 08:01, CardReader, A.cr, checkCard, Alice-card-auth");
       expect("2 08:01, KeyIssuer, A.ki, keyIssued, A.ki-1");
       input ("2 08:05, DSTouchScreen, A.ts, startReg, John");
       expect("2 08:05, CardReader, A.cr, enterCardAndPin");
       input ("2 08:06, CardReader, A.cr, checkCard, John-card-auth");
       expect("2 08:06, KeyIssuer, A.ki, keyIssued,A.ki-2");
   }
```
This is the use case for the user registration. The input device is the touch screen and takes as input the name of the new user. It prompts the user to enter their card and pin into the card reader which is an I/O device. Once card details are authorised the key issuer which is an output device issued a key with a unique keyID for that user.

HireBike()
```
public void hireBike(){
      logger.info("Starting test: hireBike");
      setupDemoSystemConfig();
      input ("2 10:20, KeyReader, A.2.kr, insertKey, A.ki-2, bike2");
      expect("2 10:20, OKLight,  A.2.ok, flashed");
      expect("2 10:20, BikeLock, A.2.bl, unlocked");
   }
```

This code is for the use case HireBike. It takes as an input the user's key which is placed in the keyReader, an input device and also takes the bikeID for the specific bike the user wants to hire out. This in return flashes a green light (output device) and unlocks the bike (output device).

ViewUserActivity()

```
public void viewUserActivity(){
      logger.info("Starting test: viewUserActivity");
      setupDemoSystemConfig();
      input ("2 19:00, DSTouchScreen, A.ts, viewActivity");
      expect ("2 19:00, DSTouchScreen, A.ts, viewPrompt, insertKey");
       input ("2 19:00, KeyReader, A.kt, keyInsertion, A.ki-2");
      expect ("2 19:00, DSTouchScreen, A.ts, viewUserActivity, HireTime, HireDS, ReturnDS,
Duration (min), 10:20:00, A, B, 160");
   }
```

This code is for the use case ViewUserActivity. Here the user presses the viewActivity button(input device) on the touchScreen. This then prompts the user to insert their key(output since it is a message), once the user inserts their key into the keyReader(input device) the KeyID is recognised and the activity for that user is displayed on the screen(output device).

showHighLowOccupancy()

```
public void showHighLowOccupancy() {
       logger.info("Starting test: showHighLowOccupancy");
       setupDemoSystemConfig();
       setupBikes();
       input ("2 08:00, Clock, clk, tick");
       input ("2 08:01, Clock, clk, tick");
       input ("2 08:02, Clock, clk, tick");
       expect("2 08:00, HubDisplay, hd, viewOccupancy, unordered-tuples, 6,"
           + "DSName, East, North, Status, #Occupied, #DPoints,"
           + " A,  500,   350,    LOW,        3,        15,"
           + " B,  400,   300,    LOW,        4,        13");
   }
```

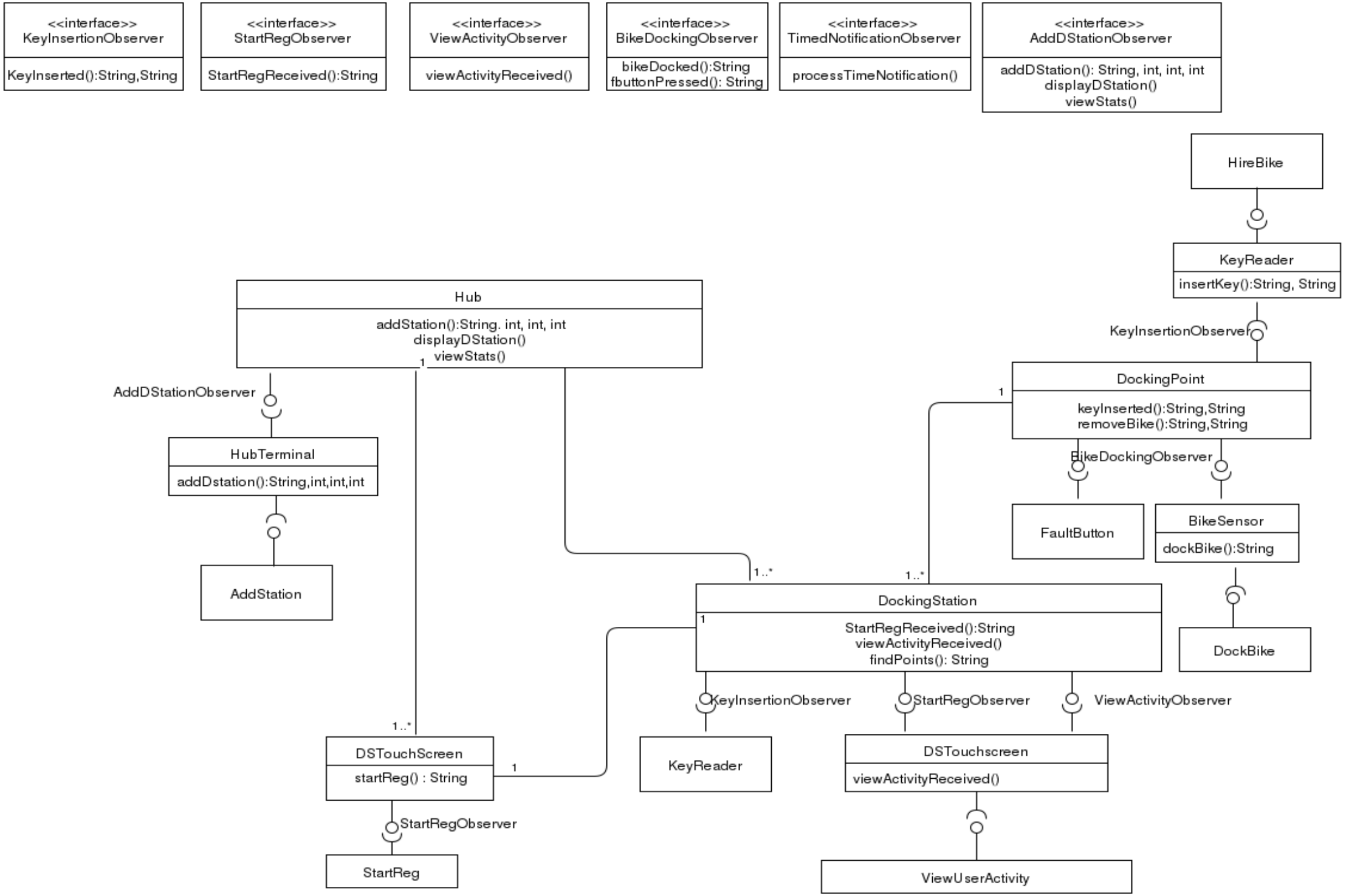
This code is the use case for ViewOccupancy. The clock at the Hub is responsible for refreshing the information for the wall display which displays the information it contains, so the information about the various docking stations.

Event Testing

Event testing is the procedure of taking two events and if they are equal then return True, else if they are not equal return False. In this project we first checked if two events with the same parameters are equal. We used assert.equals() to check the equality of two events with the same parameters. We used the asssert.NotEquals() to check if two events with different parameters are not equal. We created eight test cases where in each case we change one different parameter (so that the two events have only one different parameter between them). So, for example we have event1 and event2, where in the first case they have only a different first parameter, then in the second case they have a different second parameter, then in another case they have a different date (day or time), etc. Afterwards, we created two test cases where we check if two lists of events are equal. We firstly check if two lists of events with different sequences of same events are equal when they have the same date. This is supposed to return true. Then we check if two lists of events with different sequences of same events are equal when they don't have the same date. This is supposed to return false.

Use Case Diagram

The following diagram contains the target use cases and how they are linked so that they can perform the required actions. Beginning with the RegisterUser use case, when the new user initiates the process at the docking terminal's touch screen the information is stored in the DockingStation and the Hub is also updated as it must add a new user to its database. At the docking station when the Key reader is activated it finds the nearest available docking stations with available docking points. Still at the Docking Station, when the user requests an activity summary (viewUserActivity), the system returns the specific user's activity summary. At the docking point we have the KeyReader being activated which means a user is hiring a bike or a member of staff is removing a bike. Once a bike is hired/removed the information for that docking station is updated. Still at the Docking Point when a bike is returned by a user or a new bike is added the Bike sensor detects it being returned and so the returned bike is locked and the information for that docking station is updated. If a fault button is pressed the bike returned is set as faulty and the information for that bike is updated. Now at the Hub, we have the use case of AddStation, which is done at the HubTerminal, which once the new station has been added the Hub is updated and the new station is also displayed.


<<interface>>
KeyInsertionObserver
KeyInserted():String,String
<<interface>>
StartRegObserver
StartRegReceived():String
<<interface>>
ViewActivityObserver
viewActivityReceived()
<<interface>>
BikeDockingObserver
bikeDocked():String
fbuttonPressed(): String
<<interface>>
TimedNotificationObserver
processTimeNotification()
<<interface>>
AddDStationObserver
addDStation(): String, int, int, int
displayDStation()
viewStats()
HireBike
KeyReader
insertKey():String, String
KeyInsertionObserver
Hub
addStation():String. int, int, int
displayDStation()
viewStats()
1
AddDStationObserver
HubTerminal
addDstation():String,int,int,int
AddStation
DockingPoint
keyInserted():String,String
removeBike():String,String
BikeDockingObserver
FaultButton
BikeSensor
dockBike():String
DockBike
1..*
1..*
DockingStation
StartRegReceived():String
viewActivityReceived()
findPoints(): String
KeyInsertionObserver
StartRegObserver
ViewActivityObserver
1..*
DSTouchScreen
startReg() : String
1
KeyReader
DSTouchscreen
viewActivityReceived()
StartRegObserver
StartReg
ViewUserActivity

Tests for Supplementary Use Cases

removeBike()

```
public void removeBike(){
       logger.info("Starting test: removeBike");

      setupDemoSystemConfig();
      setupFaultyBikes();
      input ("2 13:30, KeyReader, B.2.kr, insertStaffKey, staffkey, bike2");
      expect("2 13:30, OKLight,  B.2.ok, flashed");
      expect("2 13:30, BikeLock, B.2.bl, unlocked");
          }
```

This code is for the use case RemoveBike. Staff member identifies themselves with their staffKey at the keyReader(input device) for the specific bike. The OKLight flashes(output device) and the bike is unlocked(output device).

reportFault()

```
public void reportFault(){
      logger.info("Starting test: reportFault");
      setupDemoSystemConfig();

      input("2 13:15, FaultButton, A.1.fb, pressFButton, bike2");
      expect("2 13:15, BikeLock, A.1.bl, locked");
      input("2 13:17, FaultButton, A.1.fb, pressingButton");
      expect("2 13:17, RedLight, A.1.rl, flashed");
   }
```

This code is for the use case ReportFault. Here the user returns the bike, and the system takes its unique ID just as when a normal bike is returned. The bike is locked(output device) and the user presses the fault button(input device) within two minutes. Once the fault button is pressed the red light flashes(output device) which indicates the bike is faulty.

findFreePoints()

```
 public void findFreePoints(){
       logger.info("Starting test: findFreePoints");
      setupDemoSystemConfig();
      setupBikes();
      input("2 12:05, KeyReader, A.kt, insertKey1, A.ki-2");
      expect("2 12:05, DSTouchScreen, A.ts, showLocations,B,400,300,13");
   }
```

This code is for the use case FindFreePoints. A user with the specific ID inserts their Key into the KeyReader(input device) at the docking station terminal. This tells the system to display the nearest docking stations with available docking points. TouchScreen(output device) displays the location of available docking stations.

chargeUsers()

```
public void chargeUsers(){
      logger.info("Starting test: chargeUsers");
      setupDemoSystemConfig();
      setUpSchedule();
       input ("8 00:00, Clock, clk, tick");
      expect ("8 00:00, BankServer, bb, paymentCompleted");
   }
```

This code is for the use case chargeUser. The clock at the Hub detects when it is midnight, as this is

when all the users are charged for their usage during that day. Once the clock hits midnight and each user is charged, the BankServer should return that each payment is completed.

viewTheStatus()

```
 public void viewTheStatus(){
        logger.info("Starting test: viewTheStatus");
       setupDemoSystemConfig();
       setupUsers();
       input("5 16:10, HubTerminal, ht, viewStats");
       expect("5 16:10, HubDisplay, hd, status, JourneyNumber, AvailableDP, noOfUsers,
averageJourneyTime, 2, 28 ,2, 80");
    }
```

This code is for the use case ViewStats. The operator at the Hub station chooses the viewStats option at the Hub Terminal(input device). The HubDisplay(output device) then displays all the information corresponding to the docking stations.