Report on Inf2C-SE Coursework 3

Keqi Li (s1408733) Jinhong Lu (s1569197)

February 20, 2017

Contents

How to compile and run the code

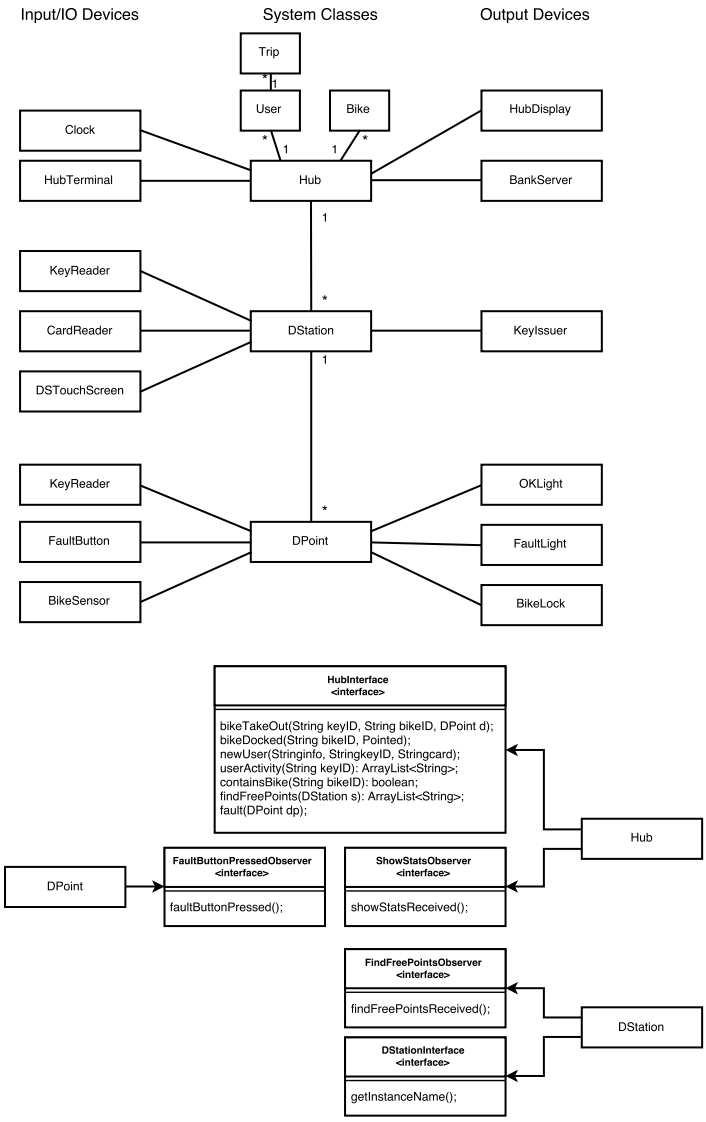
- $ export CLASSPATH=bin:lib/junit.jar:lib/org/hamcrest.core_1.3.0.v201303031735.jar
- $ javac -sourcepath src -d bin src/bikescheme/*.java
- $ java bikescheme.AllTests

How to output the output log

- $ java bikescheme.AllTests &> output.log

1 Summary of the Implementation

This design implementation of the tasks is based on the demonstration by Prof. Paul Jackson, the overall class diagram with the newly designed classes and interfaces are shown in diagram 1. Changes in the Hub class, DStation class and DPoint class are specified in Appendix.



2 Design of New System Classes

2.1 User

```
1   private String userAccount;
2   private String keyID;
3   private String card;
4   private ArrayList<Trip> trips;
5   private Trip currentTrip;
```

With the Name, KeyNumber, Card, user is able to perform use case "register user". Moreover, when a user is created, there are two information have to be joint with the user account, trips and currentTrip, in order to perform ViewUserActivity and ChargeUser

2.2 Bike

```
1   private String ID;
2   private boolean rentStatus;
3   private boolean faultStatus;
```

This class allows us to identify the bikes by BikeID and the renting status of the bikes,which allows us to observe the occupancy of each Docking Station. Meanwhile, the faultStatus provides the condition of the bike.

2.3 Trip

```
1     private String startStation, endStation;
2     private Date start, end;
3     private int day;
4     private int length;
```

This class allows us to record the information when the user hire and return the bike. Meanwhile, when user want to view their activity, this class provides information for this as well.

3 Design of new System Devices

3.1 FaultButton

```
1     public void faultBike(){
2         logger.fine(getInstanceName());
3         observer.faultButtonPressed();
4     }
```

This device implements as an input device and extends with provided code, AbstractInputDevice.

3.2 FaultLight

```
1     public void flash() {
2         logger.fine(getInstanceName());
3
4         String deviceClass = "FaultLight";
5         String deviceInstance = getInstanceName();
6         String messageName = "flashed";
7         List<String> valueList = new ArrayList<String>();
8
9         super.sendEvent(
10            new Event(
11                Clock.getInstance().getDateAndTime(),
12                deviceClass,
13                deviceInstance,
14                messageName,
15                valueList));
16
17    }
```

This device implements as an output device and extends with provided code, AbstractOutputDevice.

3.3 BankServer

```
1     public void chargeUsers(Collection<User> users, List<String>
      data) {
2         logger.fine(getInstanceName());
3
4         String deviceClass = "BankServer";
5         String deviceInstance = getInstanceName();
6         String messageName = "chargeUser";
7
8         List<String> messageArgs = new ArrayList<String>();
```

```
9           String[] preludeArgs =
10              { "unordered-tuples","5",
11                "UserName", "KeyID", "#Trips", "TimeSum", "
    DebitAmount" };
12
13          messageArgs.addAll(Arrays.asList(preludeArgs));
14          messageArgs.addAll(data);
15
16          super.sendEvent(
17                  new Event(
18                      Clock.getInstance().getDateAndTime(),
19                      deviceClass,
20                      deviceInstance,
21                      messageName,
22                      messageArgs));
23      }
```

This device implements as an output device and extends with provided code, AbstractOutputDevice

4 Design of New Interfaces

4.1 ShowStatsObserver

```
1     public void showStatsReceived();
```

This observer provides a function with use case, ShowStats, when operator request from the Hub Terminal.

4.2 FaultButtonPressedObserver

```
1     public void faultButtonPressed();
```

This observer provides a function with use case, ReportFault, when the fault button is activated and pressed by user.

4.3 FindFreePointObserver

```
1     public void findFreePointsReceived();
```

This observer provides a function with use case, FindFreePoints, when the user requests from the docking station with the DSTouchScreen.

4.4 HubInterface

```
1     public void bikeTakeOut(String keyID, String bikeID, DPoint
      d);
2
3     public void bikeDocked(String bikeID, DPoint d);
4
5     public void newUser(String info, String keyID, String card);
6
7     public ArrayList<String> userActivity(String keyID);
8
9     public boolean containsBike(String bikeID);
10
11    public ArrayList<String> findFreePoints(DStation s);
12
13    public void fault(DPoint dp);
```

This interface provides several functions that supports most the use cases that have to interact between Docking Station (include Docking Point) and Hub Terminal

4.5 DStationInterface

```
1     public String getInstanceName();
```

This interface provides the name of the docking station that this docking point that is belonged to.

5 Implementation of Use Case (System-Test)

5.1 AddDStation

For AddDStation, there are four parameters required from input to perform the functionality:

1. Instance Name - The name of the Station
2. East - The East Position of the Station
3. North - The North Position of the Station
4. DPoints - Total Number of Docking Point in the Station

```
1  @Test
2    public void addDStation() {
3        logger.info("Starting test: addStation");
4        setupSystemConfig();
5        input("3 07:00, HubTerminal, ht, addDStation, C,   200,
     300, 10");
6        input("3 07:00, HubTerminal, ht, addDStation, D,   100,
     200, 10");
7    }
```

Corresponding Requirement

1. Operator enter a valid name
2. Operator enter a valid East Position
3. Operator enter a valid North Position
4. Operator enter a valid number of Docking Point in this Docking Station

5.2 AddBike

For AddBike, there is one parameter needed to identify the bike as triggering input from bike sensor.

1. BikeID : the id of the bike that is going to add

```
@Test
  public void addBike() {
      logger.info("Starting test: addBike");
      setupSystemConfig();
      logger.info("addBike");
      input("2 09:00, BikeSensor, A.1.bs, dockBike, bike-15");
  }
```

Corresponding Requirement

1. Bike has an valid bike ID
2. Staff has put the bike into a working Docking Point
3. Sensor has successfully scanner the ID of the bike
4. Docking Station push the bike ID to the Hub Terminal

5.3 RegisterUser

1. personalInfo : the name of the user
2. card : the credit card authentication code from bank.
3. keyID : the key for user which issued by the key issuer.

```
@Test
  public void registerUser() {
      logger.info("Starting test: registerUser");
      setupSystemConfig();
      // 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");
  }
```

Corresponding Requirement

1. User enter an valid user name
2. User has insert an valid bank card and enter valid passwork for the card
3. Key Issuer has enough keys and each key with valid key ID
4. key Issuer issues the key
5. System should save the user information

5.4 HireBike

1. KeyID : The key that is used to hired the bike
2. BikeID : The bike that is hired
3. Docking Point : the station that the user hired the bike

```
@Test
  public void hireBike() {
      logger.info("Starting test: hireBike");
      registerUser();
      setupBikes();
      input("2 08:00, KeyReader, A.2.kr, insertKey, A.ki-1");
      expect("2 08:00, BikeLock, A.2.bl, unlocked");
      expect("2 08:00, OKLight, A.2.ok, flashed");
  }
```

Corresponding Requirement

1. There is a bike occupying the docking point
2. User has inserted the valid key
3. KeyReader has verified the key with Hub Terminal
4. BikeLock should unlock the bike
5. OKLight should flash
6. System should record the bike status is rent

5.5 ReturnBike

1. BikeID : the bike that is going to be returned
2. Docking Point : the station that the user returned the bike

```
@Test
  public void returnBike() {
      logger.info("Starting test: returnBike");
      hireBike();
      input("2 09:00, BikeSensor, A.2.bs, dockBike, bike-2");
      expect("2 09:00, OKLight, A.2.ok, flashed");
      expect("2 09:00, BikeLock, A.2.bl, locked");
  }
```

Corresponding Requirement

1. There is a record that this bike is rent out by a user
2. Sensor should verify the bike ID that this bike is belonged to us
3. OKLight should flash
4. BikeLock should lock the bike
5. System should record the bike status is returned

5.6 ViewUserActivity

1. HireTime : The time that user start to hire the bike
2. HireDS : The docking station that user hire the bike
3. ReturnDS : The docking station that user return the bike
4. Duration(min) : The time that user hire for the bike

```
@Test
  public void viewUserActivity() {
      logger.info("Starting test: viewUserActivity");
      returnBike();
      logger.info("viewUserActivity");
      input("2 18:00, DSTouchScreen, A.ts, viewActivity");
```

```
7         expect("2 18:00, DSTouchScreen, A.ts, viewPrompt, PLEASE
      INSERT YOUR KEY.");
8         input("2 18:01, KeyReader, A.kr, keyInsertion, A.ki-1");
9         expect("2 18:01, DSTouchScreen, A.ts, viewUserActivity,
      ordered-tuples, 4,"
10                + "HireTime, HireDS, ReturnDS, Duration (min),"
11                + "   08:00,      A,         A,           60" );
12    }
```

Corresponding Requirement

1. The user has registered with the system
2. The key is an valid key
3. Touch Screen should working fine
4. KeyReader read the key
5. Touch Screen displays information

5.7 ViewOccupancy

1. Instance Name : The name of the Station
2. East : The East Position of the Station
3. North : The North Position of the Station
4. Status : The Occupancy of the Station (High: More than 85% of Docking Point is occupied. Low: Less than 15% of Docking Point is occupied.)
5. Occupied : The number of bike in the Station
6. DPoints : Total Number of Docking Point in the Station

```
1   @Test
2     public void viewOccupancy() {
3         logger.info("Starting test: viewOccupancy");
4         returnBike();
5         input("2 18:00, Clock, clk, tick");
6         input("2 18:01, Clock, clk, tick");
7         input("2 18:02, Clock, clk, tick");
```

```
8         expect("2 18:00, HubDisplay, hd, viewOccupancy,
  unordered-tuples, 6,"
9                 + "DSName, East, North, Status, #Occupied, #
  DPoints,"
10                + "      A,    0,    0,      OK,         2,
   10,"
11                + "      B,  400,  300,    HIGH,         9,
   10");
12    }
```

Corresponding Requirement

1. The clock is working fine
2. Every 5 minutes activate the system to pull data from all stations
3. HubDisplay displays the data

6 Event-Test

6.1 assertEquals

This event test aims to test the equality of all the parameters is equal

```
1  @Test
2    public void testEqual() {
3        assertEquals(
4                new Event("1 00:00,C,i,m, a, b, c, d"),
5                new Event("1 00:00,C,i,m, a, b, c, d") );
6    }
```

6.2 assertNotEquals

This event test aims to test the unequality of all the parameters with only one parameter from each event is different

```
1  @Test
2    public void testParam1() {
3        assertNotEquals(
4                new Event("1 00:00,C,i,m, a, b, c, d"),
5                new Event("1 00:00,C,i,m, a, b, c, e") );
6    }
7
8    @Test
9    public void testParam2() {
10       assertNotEquals(
11               new Event("1 00:00,C,i,m, a, b, c, d"),
12               new Event("1 00:00,C,i,m, a, b, e, d") );
13   }
14
15   @Test
16   public void testParam3() {
17       assertNotEquals(
18               new Event("1 00:00,C,i,m, a, b, c, d"),
19               new Event("1 00:00,C,i,m, a, e, c, d") );
20   }
21
22   @Test
23   public void testParam4() {
24       assertNotEquals(
25               new Event("1 00:00,C,i,m, a, b, c, d"),
26               new Event("1 00:00,C,i,m, e, b, c, d") );
27   }
```

```
28
29      @Test
30      public void testParam5() {
31          assertNotEquals(
32                  new Event("1 00:00,C,i,m, a, b, c, d"),
33                  new Event("1 00:00,C,i,e, a, b, c, d") );
34      }
35
36      @Test
37      public void testParam6() {
38          assertNotEquals(
39                  new Event("1 00:00,C,i,m, a, b, c, d"),
40                  new Event("1 00:00,C,e,m, a, b, c, d") );
41      }
42
43      @Test
44      public void testParam7() {
45          assertNotEquals(
46                  new Event("1 00:00,C,i,m, a, b, c, d"),
47                  new Event("1 00:00,A,i,m, a, b, c, d") );
48      }
49
50      @Test
51      public void testParam8() {
52          assertNotEquals(
53                  new Event("1 00:00,C,i,m, a, b, c, d"),
54                  new Event("1 00:01,C,i,m, a, b, c, d") );
55      }
56
57      @Test
58      public void testParam9() {
59          assertNotEquals(
60                  new Event("1 00:00,C,i,m, a, b, c, d"),
61                  new Event("2 00:00,C,i,m, a, b, c, d") );
62      }
```

6.3 ListEqual

This test aims to test the list of events with same date, but different in sequences. In our project, we have tested in two way, one is to test in the same date and another is to test in different date.

```
1   @Test
2     public void testListSameDay() {
```

```
3           ArrayList<Event> events1 = new ArrayList<Event>();
4           ArrayList<Event> events2 = new ArrayList<Event>();
5
6           events1.add(new Event("1 00:00,C,i,m, a, b, c, d"));
7           events1.add(new Event("1 00:00,A,o,m, a, b, c, d"));
8
9           events2.add(new Event("1 00:00,A,o,m, a, b, c, d"));
10          events2.add(new Event("1 00:00,C,i,m, a, b, c, d"));
11
12          if (Event.listEqual(events1, events2) == false) {
13              fail("Should have returned true.");
14          }
15      }
16      @Test
17      public void testListNotSameDay() {
18          ArrayList<Event> events1 = new ArrayList<Event>();
19          ArrayList<Event> events2 = new ArrayList<Event>();
20
21          events1.add(new Event("1 00:00,C,i,m, a, b, c, d"));
22          events1.add(new Event("2 18:00,A,o,m, a, b, c, d"));
23
24          events2.add(new Event("2 18:00,A,o,m, a, b, c, d"));
25          events2.add(new Event("1 00:00,C,i,m, a, b, c, d"));
26
27          if (Event.listEqual(events1, events2) == true) {
28              fail("Should have returned false.");
29          }
```

7 Supplementary Task of Use Case

The following is the implementation of a supplementary task of use cases. The newly designed classes and interfaces are included in the figure on page 3.

7.1 RemoveBike

1. KeyID : The ID is admin
2. BikeID : The bike that is going to be removed
3. Docking Point : The Docking Point that the bike belong to

```
1  @Test
2    public void removeBike() {
3        logger.info("Starting test: removeBike");
4        setupSystemConfig();
5        setupBikes();
6        logger.info("removeBikes");
7        input("3 15:00, KeyReader, A.3.kr, insertKey, admin");
8        expect("3 15:00, BikeLock, A.3.bl, unlocked");
9        expect("3 15:00, OKLight, A.3.ok, flashed");
10   }
```

Corresponding Requirement

1. There is a bike occupying the docking point
2. The key ID is admin
3. KeyReader read the key
4. BikeLock should unlock the bike
5. OKLight should flash
6. System should remove the bike ID

7.2 ReportFault

1. BikeID : The bike that is fault
2. Docking Point : The docking point that the bike located

```
@Test
  public void reportFault() {
      logger.info("Starting test: reportFault");
      returnBike();
      logger.info("reportFault");
      input("2 09:01, FaultButton, A.2.fb, pressed");
      expect("2 09:01, FaultLight, A.2.fl, flashed");
  }
```

Corresponding Requirement

1. There is a bike returned at the moment
2. FaultButton is activated for 2 minutes
3. FaultButton is pressed
4. FaultLight should flash

7.3 FindFreePoints

1. Instance Name : The name of the Station
2. East : The East Position of the Station
3. North : The North Position of the Station
4. Status : The Occupancy of the Station (High: More than 85% of Docking Point is occupied. Low: Less than 15% of Docking Point is occupied.)
5. Occupied : The number of bike in the Station
6. DPoints : Total Number of Docking Point in the Station

```
@Test
  public void findFreePoints() {
      logger.info("Starting test: findFreePoints");
      setupSystemConfig();
      setupBikes();
      logger.info("findFreePoints");
      input("3 15:00, DSTouchScreen, B.ts, findFreePoints");
      expect("3 15:00, DSTouchScreen, B.ts, findFreePoints,
 unordered-tuples, 6,"
              + "DSName, East, North, Status, #Occupied, #
 DPoints,"
              + "      A, -400,  -300,     OK,          2,
  10,"
              + "      B,    0,     0,   HIGH,          9,
  10");
  }
```

Corresponding Requirement

1. Touch Screen is fully functioning
2. User requests to look for free points in surrounding area
3. Docking Station request to pull the data from Hub Terminal
4. Touch Screen displays data

7.4 ChargeUser

1. UserName : The name of the user
2. KeyID: The key id that belonged to the user
3. Trip: the number of trips that the user made
4. TimeSum: the total time of the trips that the user rent the bike
5. DebitAmount: the total amount that charge the user

```
@Test
  public void chargeUser() {
     logger.info("Starting test: chargeUser");
     returnBike();
     input("2 23:59, Clock, clk, tick");
```

```
6         expect("2 23:59, BankServer, bsv, chargeUser, unordered-
      tuples, 5,"
7                 +"UserName, KeyID, #Trips, TimeSum, DebitAmount,"
8                 +"Alice, A.ki-1,       1,       60,             3");
9     }
```

Corresponding Requirement

1. Clock is synchronised
2. System requests charge user function at 23:59
3. Calculate the user amount of charge
4. BankServer is fully functioning
5. The charge is done

7.5 ViewStats

1. Property: The event that happened in the Stations
2. Value/Stat: The number of event that happened in the Stations

```
1   @Test
2     public void viewStats() {
3         logger.info("Starting test: viewStats");
4         reportFault();
5         input("2 18:00, HubTerminal, ht, viewStats");
6         expect("2 18:00, HubTerminal, ht, showStats, ordered-
      tuples, 2,"
7                 +"Property, Value/Stat,"
8                 +"#Journeys Today, 1,"
9                 +"#Users Registered, 1,"
10                +"Average Journey Time, 60.0");
11    }
```

Corresponding Requirement

1. Docking Stations are fully functioning
2. Docking Stations record down all the events happened
3. Hub pulls data from all the stations
4. Hub displays the data

8 Appendix A - Changes in Hub

8.1 Parameter in Hub

```
public static final Logger logger = Logger.getLogger("
 bikescheme");
  private HubTerminal terminal;
  private HubDisplay display;
  private Map<String, DStation> dockingStationMap;
  // Create a map that links user to a specific key <String
 keyID, User user>
  private Map<String, User> keyToUserMap;
  // Create a map that links bike to a specific key <String
 bikeID, String
  // keyID>
  private Map<String, String> bikeToKeyMap;
  // Create a map that links a bike to a specific key <String
 bikeID, String
  // keyID>
  private Map<String, Bike> bikeMap;
  // Create an arraylist of fault bike;
  private ArrayList<Bike> faultBikes;
  private BankServer bankServer;
  public static Date chargeStart = new Date((24 * 60 - 1) * 60
  * 1000L);
```

8.2 Constructor in Hub

```
public Hub() {
      Clock.createInstance();
      // Construct and make connections with interface devices
      terminal = new HubTerminal("ht");
      terminal.setObserver(this);
      terminal.setShowStatsObserver(this);
      display = new HubDisplay("hd");
      dockingStationMap = new HashMap<String, DStation>();
      bikeMap = new HashMap<String, Bike>();
      bikeToKeyMap = new HashMap<String, String>();
      keyToUserMap = new HashMap<String, User>();
      faultBikes = new ArrayList<Bike>();
      bankServer = new BankServer("bsv");
      // Default user, staff, who use keys identified as "
 admin".
      // "****" indicates no card detail;
      newUser("STAFF", "admin", "****");
```

```
17          // Schedule timed notification for generating updates of
18          // hub display.
19          // Every 5 mins it shows the occupancy on display.
20          Clock.getInstance().scheduleNotification(new
    TimedNotificationObserver() {
21              /**
22               * Generate display of station occupancy data.
23               */
24              @Override
25              public void processTimedNotification() {
26                  logger.fine("Refreshing Display");
27                  ArrayList<String> occupancyList = showOccupancy
    (0, 0);
28                  display.showOccupancy(occupancyList);
29              }
30          }, Clock.getStartDate(), 0, 5);
31          // Schedule timed notification for charging users.
32          // Every 24 hrs it charges the users.
33          Clock.getInstance().scheduleNotification(new
    TimedNotificationObserver() {
34              @Override
35              public void processTimedNotification() {
36                  logger.fine("Charging User");
37                  bankServer.chargeUsers(keyToUserMap.values(),
    computeDebt(keyToUserMap.values()));
38              }
39          }, chargeStart, 24, 0);
40      }
```

8.3 Functions in Hub

showOccupancy – Corporate with showOccupancy and findFreePoints Use Case

```
1   public ArrayList<String> showOccupancy(int east, int north){
2     ArrayList<String> occupancyList = new ArrayList<String>();
3     // "DSName","East","North","Status","#Occupied","#DPoints"
4     for (DStation a : dockingStationMap.values()) {
5         float occupancy = ((float) a.getOccupiedDPoints() / a.
    getNoDPoints());
6         String status = "OK";
7             if (occupancy >= 0.85)
8                 status = "HIGH";
9             if (occupancy <= 0.15)
10                status = "LOW";
11            occupancyList.add(a.getInstanceName());
```

```
12              occupancyList.add(Integer.toString(a.getEastPos()-
    east));
13              occupancyList.add(Integer.toString(a.getNorthPos()-
    north));
14              occupancyList.add(status);
15              occupancyList.add(Integer.toString(a.
    getOccupiedDPoints()));
16              occupancyList.add(Integer.toString(a.getNoDPoints())
    );
17      }
18      return occupancyList;
19      }
20      }
```

findFreePoints – Corporate with findFreePoints Use Case

```
1   public ArrayList<String> findFreePoints(DStation s){
2           logger.fine("");
3           return showOccupancy(s.getEastPos(), s.getNorthPos());
4       }
```

addDStation – Corporate with addDStation Use Case

```
1   public void addDStation(String instanceName, int eastPos, int
     northPos, int numPoints) {
2           logger.fine("");
3           DStation newDStation = new DStation(instanceName,
    eastPos, northPos, numPoints);
4           dockingStationMap.put(instanceName, newDStation);
5           // Now connect up DStation to event distributor and
    collector.
6           EventDistributor d = terminal.getDistributor();
7           EventCollector c = display.getCollector();
8           newDStation.setDistributor(d);
9           newDStation.setCollector(c);
10          newDStation.setHub(this);
11          ;
12      }
```

bikeTakeOut – Corporate with RemoveBike and HireBike Use Case

```
1   public void bikeTakeOut(String keyID, String bikeID, DPoint d)
     {
2           logger.fine("");
3           if (this.keyToUserMap.containsKey(keyID)){
4               bikeToKeyMap.put(bikeID, keyID);
5               logger.fine(bikeID + " is linked to " + bikeToKeyMap
    .get(bikeID));
```

```
            bikeMap.get(bikeID).rentBike();
            keyToUserMap.get(keyID).startTrip(d.getStationName()
    );
        } else if (keyID == "admin") {
            logger.fine(bikeID + " is removed.");
            bikeMap.remove(bikeID);
        } else {
            logger.warning(keyID + " Is Not A Valid Key!");
        }
    }
```

bikeDocked – Corporate with Returnbike and AddBike

```
  public void bikeDocked(String bikeID, DPoint d) {
        logger.fine("");
        if (!this.bikeMap.containsKey(bikeID)) {
            Bike newBike = new Bike(bikeID);
            bikeMap.put(bikeID, newBike);
        } else {
            bikeMap.get(bikeID).returnBike();
            logger.fine(bikeID); keyToUserMap.get(bikeToKeyMap.
    get(bikeID)).endTrip(d.getStationName());
            bikeToKeyMap.remove(bikeID);
        }
    }
```

newUser – Corporate with RegisterUser Use Case

```
  public void newUser(String info, String keyID, String card) {
        User u = new User(info, keyID, card);
        logger.fine(keyID);
        this.keyToUserMap.put(keyID, u);
    }
```

userActivity – Corporate with ViewUserActivity Use Case

```
  public ArrayList<String> userActivity(String keyID) {
        ArrayList<String> ac = new ArrayList<String>();
        User u = this.keyToUserMap.get(keyID);
        logger.fine(keyToUserMap.get(keyID).getKeyNumber());
        for (Trip t : u.getTrips()) {
            ac.add(Clock.format(t.getStart()).split(" ")[1]);
            ac.add(t.getStartStation());
            ac.add(t.getEndStation());
            ac.add(Integer.toString(t.getLength()));
        }
        return ac;
    }
```

showStatesReceived – Corporate with ViewStats Use Case

```
1  public void showStatsReceived(){
2        logger.fine("");
3        ArrayList<String> stat = new ArrayList<String>();
4        stat.add("Property");
5        stat.add("Value/Stat");
6        stat.add("#Journeys Today");
7        int journeyNo = 0, journeyTimeSum = 0;
8        for (User u : keyToUserMap.values()) {
9            journeyNo += u.getTrips().size();
10           for (Trip t : u.getTrips()) {
11               journeyTimeSum += t.getLength();
12           }
13       }
14       stat.add(Integer.toString(journeyNo));
15       stat.add("#Users Registered");
16       stat.add(Integer.toString(keyToUserMap.size()));
17       stat.add("Average Journey Time");
18       stat.add(Float.toString((float) journeyTimeSum/journeyNo
     ));
19       terminal.showStats(stat);
20    }
```

containsBike – To check is the bikeID inside the BikeMap

```
1  public boolean containsBike(String bikeID) {
2        if (bikeMap.containsKey(bikeID))
3            return true;
4        else
5            return false;
6     }
```

fault – to add the bikeID to the faultBikes List

```
1  public void fault(DPoint dp) {
2        logger.fine("");
3        faultBikes.add(bikeMap.get(dp.getBikeID()));
4     }
```

computeDebt – To compute the amount of money that going to charge the users and corporate with Charge User Use Case

```
1  public ArrayList<String> computeDebt(Collection<User> users) {
2        // Computing each user's debt amount.
3        ArrayList<String> debtData = new ArrayList<String>();
4        for (User user : users) {
5            int debt = 0, timeSum = 0;
```

```
            // 123456
            for (Trip t : user.getTrips()) {
                int time = t.getLength();
                timeSum += time;
                if (time > 0) {
                    debt += 1;
                    if (time > 30) {
                        if ((time - 30) % 30 > 0)
                            debt += ((time - 30) / 30) * 2;
                        else
                            debt += ((time - 30) / 30) * 2;
                    }
                }
            }
            // Add to print out list
            if (debt > 0) {
                //"UserName", "KeyID", "#Trips", "TimeSum", "
    DebitAmount"
                String[] userDebt =
                        {user.getUserAccount(),
                         user.getKeyNumber(),
        Integer.toString(user.getTrips().size()),
                         Integer.toString(timeSum),
                         Integer.toString(debt)};
                debtData.addAll(Arrays.asList(userDebt));
            }
        }
        return debtData;
    }
```

getDStation – to return the Docking Station Name

```
  public DStation getDStation(String instanceName) {
        return dockingStationMap.get(instanceName);
    }
```

9 Appendix B - Changes in DStation

9.1 Parameter in DStation

```
public static final Logger logger = Logger.getLogger("
 bikescheme");
  private String instanceName;
  private int eastPos;
  private int northPos;
  private DSTouchScreen touchScreen;
  private CardReader cardReader;
  private KeyIssuer keyIssuer;
  private List<DPoint> dockingPoints;
  private KeyReader keyReader;
  private HubInterface hub;
```

9.2 Constructor in DStation

```
public DStation(
          String instanceName,
          int eastPos,
          int northPos,
          int numPoints) {
   // Construct and make connections with interface devices
      this.instanceName = instanceName;
      this.eastPos = eastPos;
      this.northPos = northPos;
      touchScreen = new DSTouchScreen(instanceName + ".ts");
      touchScreen.setObserver(this);
      // Set observers to recognise "view activity" and "find
 free points" options
      touchScreen.setViewActivityObserver(this);
      touchScreen.setFindFreePointsObserver(this);
      cardReader = new CardReader(instanceName + ".cr");
      keyIssuer = new KeyIssuer(instanceName + ".ki");
      keyReader = new KeyReader(instanceName + ".kr");
      dockingPoints = new ArrayList<DPoint>();
      for (int i = 1; i <= numPoints; i++) {
          DPoint dp = new DPoint(instanceName + "." + i, i -
 1);
          dockingPoints.add(dp);
      }
  }
```

9.3 Functions in DStation

startRegReceived – Corporate with RegisterUser Use Case

```
1  public void startRegReceived(String personalInfo) {
2        logger.fine("Starting on instance " + getInstanceName())
     ;
3        cardReader.requestCard();  // Generate output event
4        logger.fine("At position 1 on instance " +
     getInstanceName());
5        String card = cardReader.checkCard();    // Pull in non-
     triggering input event
6        logger.fine("At position 2 on instance " +
     getInstanceName());
7        String keyID = keyIssuer.issueKey(); // Generate output
     event
8        // Create a new User class in the Hub system and links
     it to the key.
9        hub.newUser(personalInfo, keyID, card);
10    }
```

setHub – To set connection between the Docking Station and the Hub

```
1  public void setHub(HubInterface h){
2        this.hub = h;
3        for (DPoint dp : dockingPoints) {
4            dp.setInterface(this, hub);
5        }
6    }
```

viewActivityReceived – Corporate with viewUserActivity Use Case

```
1  public void viewActivityReceived(){
2        logger.fine(getInstanceName());
3        touchScreen.showPrompt("PLEASE INSERT YOUR KEY.");
4        String keyID = keyReader.waitForKeyInsertion();
5        ArrayList<String> userInfoData;
6        userInfoData = hub.userActivity(keyID);
7        touchScreen.showUserActivity(userInfoData);;
8    }
```

findFreePointsReceived – Corporate with findFreePoints Use Case

```
1  public void findFreePointsReceived(){
2        logger.fine(getInstanceName());
3        ArrayList<String> occupancyList;
4        occupancyList = hub.findFreePoints(this);
5        touchScreen.showFreePoints(occupancyList);
6    }
```

getNoDPoints – To get the number of Docking Point in this docking station

```
1   //Return number of DPoints Modified
2     public int getNoDPoints(){
3         return dockingPoints.size();
4     }
```

getOccupiedDPoints – To get the number of occupied docking points in the docking station

```
1   //Return number of occupied DPoints Modified
2     public int getOccupiedDPoints(){
3         int occupiedNumber = 0;
4         for (DPoint p : dockingPoints){
5             if (p.isOccupied()) occupiedNumber++;
6         }
7         return occupiedNumber;
8     }
```

10 Appendix C - Changes in DPoint

10.1 Parameter in DPoint

```
public static final Logger logger = Logger.getLogger("
 bikescheme");
  private KeyReader keyReader;
  private OKLight okLight;
  private BikeSensor bikeSensor;
  private BikeLock bikeLock;
  private String instanceName;
  private int index;
  private boolean occupiedStatus;
  private String bikeID;
  private FaultButton faultButton;
  private Date lastDocked;
  private FaultLight faultLight;
  private HubInterface hub;
  private DStationInterface station;
```

10.2 Constructor in DPoint

```
public DPoint(String instanceName, int index) {
      // Construct and make connections with interface devices
      keyReader = new KeyReader(instanceName + ".kr");
      keyReader.setObserver(this);
      okLight = new OKLight(instanceName + ".ok");
      // NEW PARAMETERS
      bikeSensor = new BikeSensor(instanceName + ".bs");
      bikeSensor.setObserver(this);
      bikeLock = new BikeLock(instanceName + ".bl");
      occupiedStatus = false;
      faultButton = new FaultButton(instanceName + ".fb");
      faultButton.setObserver(this);
      faultLight = new FaultLight(instanceName + ".fl");
      this.instanceName = instanceName;
      this.index = index;
  }
```

10.3 Functions in DPoint

keyInserted – Corporate with HireBike, AddBike, RemoveBike Use Case

```
public void keyInserted(String keyId) {
      logger.fine(getInstanceName());
```

```
        bikeLock.unlock();
        okLight.flash();
        hub.bikeTakeOut(keyId, bikeID, this);
        this.occupiedStatus = false;
    }
```

bikeDocked – Corporate with ReturnBike Use Case

```
  public void bikeDocked(String bikeID) {
        logger.fine(getInstanceName());
        if (hub.containsBike(bikeID)){
            okLight.flash();
            bikeLock.lock();
        }
        hub.bikeDocked(bikeID, this);
        this.bikeID = bikeID;
        this.occupiedStatus = true;
        this.lastDocked = Clock.getInstance().getDateAndTime();
    }
```

faultButtonPressed – Corporate with FaultBike Use Case

```
  public void faultButtonPressed(){
        logger.fine(getInstanceName());
        if (Clock.minutesBetween(lastDocked, Clock.getInstance()
   .getDateAndTime())<=2) {
            faultLight.flash();
            hub.fault(this);
        }
    }
```

isOccupied – To return value that is there a bike or not

```
  public boolean isOccupied() {
        return this.occupiedStatus;
    }
```

getStationName – To return the name of the docking station that the docking point that is belonged to

```
  public String getStationName() {
        return station.getInstanceName();
    }
```

getBikeID – To return the bikeID of the bike inserted, or occupied

```
  public String getBikeID() {
        return bikeID;
    }
```