Inf2C Software Engineering 2015-16

Coursework 2
Creating a software design for a
city bike-hire scheme

Andra Zaharia (s1402967)
Ramona Comanescu (s1427590)

November 10, 2015

2.1. Introduction

The aim of this document is to create a design, using UML class and sequence diagrams for a new self-serve hire-bike scheme in Edinburgh.

The CyclED system consists of several docking stations placed at strategic positions across town and an Operations Hub which monitors all the activity happening in the docking stations. It facilitates the supervision, assistance and maintenance of the entire process of hiring bikes around the city, while keeping track of all states of the docking stations and any changes made.

For further information on the details of the system you can refer to the following documentation: coursework1 and coursework2.

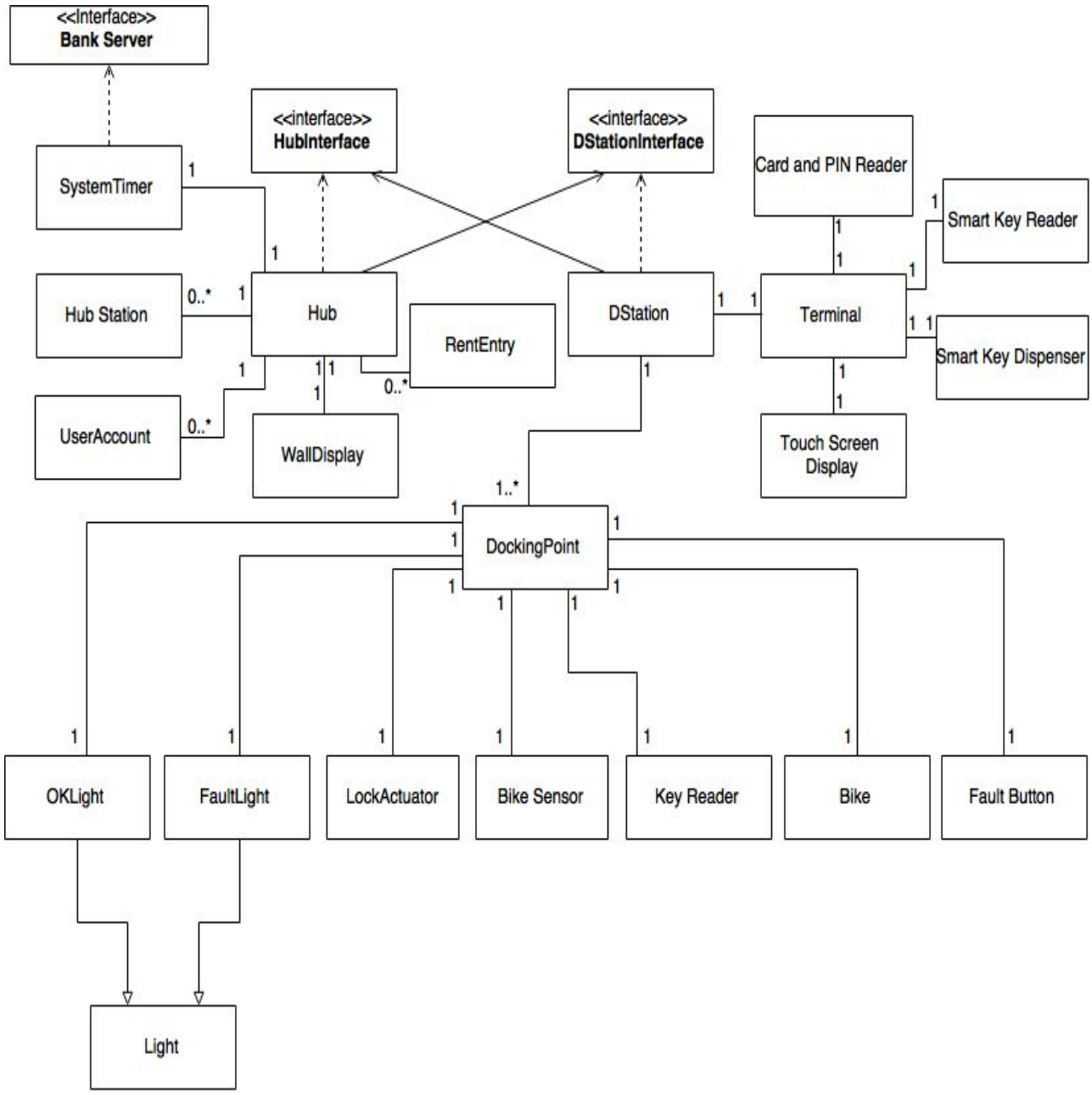
2.2 Design Assumptions

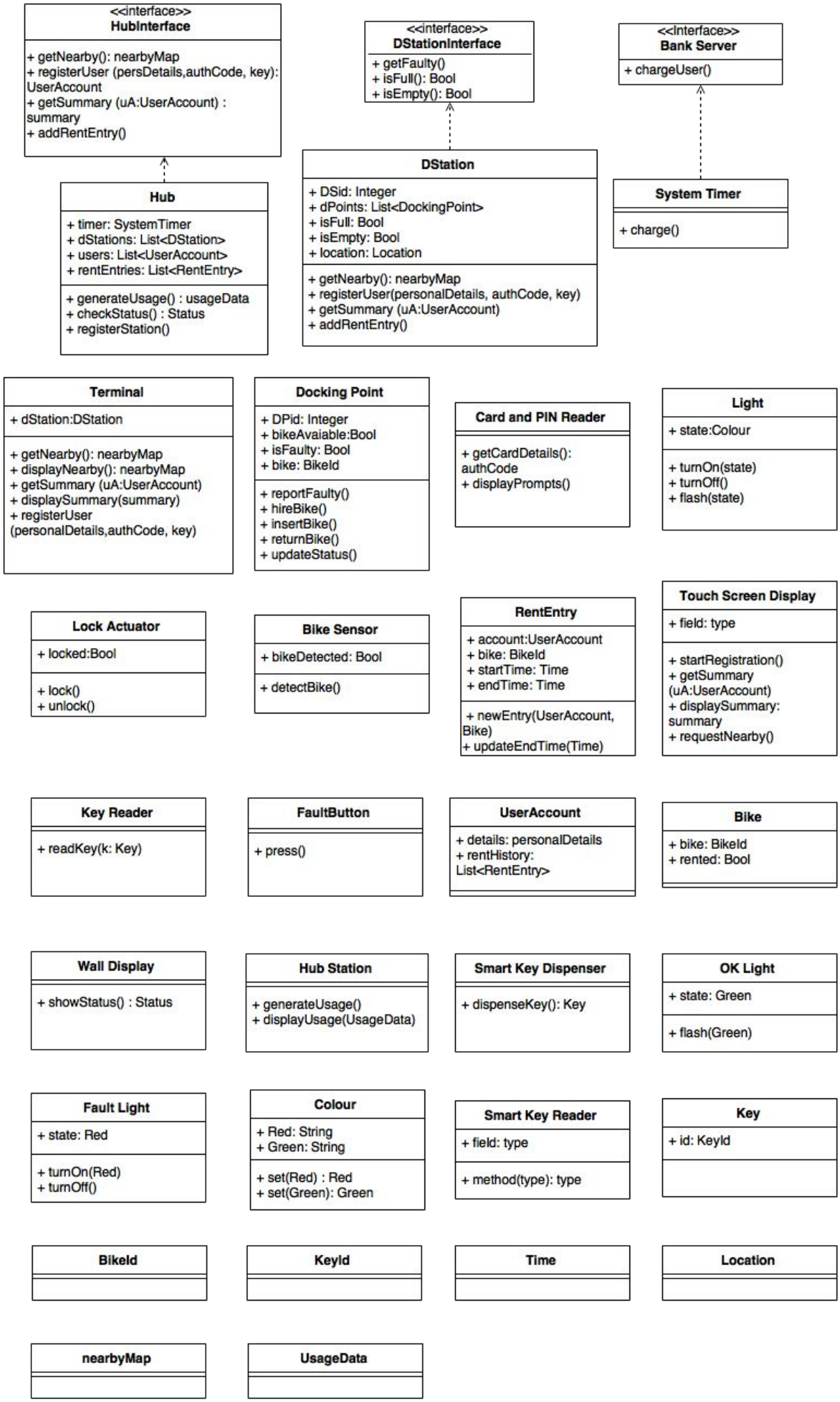
1. Wireless data-links as input or output methods between Hub Station and Docking Stations have not been included for simplicity.
2. Turning the green light on and off as an output method is abstracted into a single output event modelled by the flash() operation.
3. At a docking point key reader, the key insertion sensor, key data reader and key removal sensor is abstracted into a single class operation, namely readKey() input event.
4. It is assumed that after the insertion of the key, the user removes it immediately.
5. The Button class is abstracted to not provide a release() operation, as the system is assumed to only handle a press() input event.
6. Touch-screen of the docking station terminal is modelled as a single input/output device.
7. Start registration and Enter personal details operations at an available terminal are merged into a single operation, namely sStartRegistration, that returns the user's personal details.
8. The operation of displaying docking point availability at nearby docking stations assumes there is a way to output some kind of a map on screen(abstracted by the nearbyMap class).
9. The hub operator working station as a whole models the keyboard, mouse and monitor.
10. The card and PIN reader has an operation getCardPinInfo that combines prompting for input and output information.

11. The get card and PIN info returns either an authorisation code or some token indicating a problem.
12. It is assumed that the card and PIN reader device is connected to a banking server wirelessly, so that it checks the card validity when inserted.
13. There is a class for the banking server interface supporting a charge operation.
14. The charge operation is assumed to automatically start at midnight, going to the list of RentEntries stores by the hub.
15. The exact format in which data such as Usage Reports and Account History is presented is left for implementation.
16. It is assumed that the activSummary() operation implies both the user inserting some login details using the touch screen display and then requesting a summary for that account.
17. Classes like BikeId and ClassId have been abstracted to represent some kind of unique identification (instead of simply using a String or Integer).

2.3. Static model

2.3.1. UML class model




<<interface>>
HubInterface
+ getNearby(): nearbyMap
+ registerUser (persDetails,authCode, key): UserAccount
+ getSummary (uA:UserAccount) : summary
+ addRentEntry()
<<interface>>
DStationInterface
+ getFaulty()
+ isFull(): Bool
+ isEmpty(): Bool
<<Interface>>
Bank Server
+ chargeUser()
Hub
+ timer: SystemTimer
+ dStations: List<DStation>
+ users: List<UserAccount>
+ rentEntries: List<RentEntry>
+ generateUsage() : usageData
+ checkStatus() : Status
+ registerStation()
DStation
+ DSid: Integer
+ dPoints: List<DockingPoint>
+ isFull: Bool
+ isEmpty: Bool
+ location: Location
+ getNearby(): nearbyMap
+ registerUser(personalDetails, authCode, key)
+ getSummary (uA:UserAccount)
+ addRentEntry()
System Timer
+ charge()
Terminal
+ dStation:DStation
+ getNearby(): nearbyMap
+ displayNearby(): nearbyMap
+ getSummary (uA:UserAccount)
+ displaySummary(summary)
+ registerUser (personalDetails,authCode, key)
Docking Point
+ DPid: Integer
+ bikeAvaiable:Bool
+ isFaulty: Bool
+ bike: BikeId
+ reportFaulty()
+ hireBike()
+ insertBike()
+ returnBike()
+ updateStatus()
Card and PIN Reader
+ getCardDetails(): authCode
+ displayPrompts()
Light
+ state:Colour
+ turnOn(state)
+ turnOff()
+ flash(state)
Lock Actuator
+ locked:Bool
+ lock()
+ unlock()
Bike Sensor
+ bikeDetected: Bool
+ detectBike()
RentEntry
+ account:UserAccount
+ bike: BikeId
+ startTime: Time
+ endTime: Time
+ newEntry(UserAccount, Bike)
+ updateEndTime(Time)
Touch Screen Display
+ field: type
+ startRegistration()
+ getSummary (uA:UserAccount)
+ displaySummary: summary
+ requestNearby()
Key Reader
+ readKey(k: Key)
FaultButton
+ press()
UserAccount
+ details: personalDetails
+ rentHistory: List<RentEntry>
Bike
+ bike: BikeId
+ rented: Bool
Wall Display
+ showStatus() : Status
Hub Station
+ generateUsage()
+ displayUsage(UsageData)
Smart Key Dispenser
+ dispenseKey(): Key
OK Light
+ state: Green
+ flash(Green)
Fault Light
+ state: Red
+ turnOn(Red)
+ turnOff()
Colour
+ Red: String
+ Green: String
+ set(Red) : Red
+ set(Green): Green
Smart Key Reader
+ field: type
+ method(type): type
Key
+ id: KeyId
BikeId
KeyId
Time
Location
nearbyMap
UsageData

2.3.2. High-level Description

Our design aims to follow good design principles: low coupling between components and high cohesion within components.
To begin with, we made sure there is low coupling between components by using interfaces for the Docking Station and Hub classes, thus allowing us to later change the actual implementation without affecting the provided operations.
Our design flow is based on levels of functionality. Therefore, the docking station has a list of its docking points and can get messages from them, further calling functions provided by the Hub to store/update information. Likewise, the Hub holds a list of the docking stations and can get statuses for each of them. We also use the Hub to store the UserAccounts and the RentEntries, so we can store the hiring operations and update on return.
When an user makes a request, we pass the call through the Terminal and then the Docking station, the actual data being obtained through the Hub interface; respective operations hold the same names to make it easier to understand, but their main purpose is to only pass calls further to the hub. We only provide access in this direction- i.e. there is no direct link from a terminal to the hub, making it easier to change low level components.
At registration, a new UserAccount object is created, storing the user and card details, and also the key that was used.
When a new hiring takes places, we pass information from the terminal to the docking station, which in turn calls the hub to create a RentEntry entity, which holds the userAccount(which also stores the key) and the bikeId.
Similarly, the Terminal controls all the devices associated with it - the PIN and Card Reader, the touch screen, the Smart Key Reader and the Smart Key Dispenser; in this way, we ensure we have high cohesion, passing all the actions through the terminal.
In the design presented, we have constructed classes for each of the input and output methods, in order to clearly present the system's components. The UML class diagram clearly shows the relations between all classes, as well as their attributes and operations, which makes it easier to implement and reduces the amount of uncertainty when building the system.
However, the design has certain abstractions, all of which are presented in the Design Assumptions section (see 2.2). The abstractions were made to present a simpler model than the realistic model, while still showing a fair amount of detail to understand the system. This implies that most of the input and output sequence events that might occur together at a single device are modelled into pairs of events. Hence the system has a single thread of control.
We believe that we have created a functional design that meets the requirements.

2.3.3. Further class documentation

Hub class.

The Hub is the class that centralizes all information: hence, it keeps a list of userAccounts and a list of RentEntries, which it can then use to generate the required reports. It also holds a list of dStations and can use the dStation interface to get real time information.

Attributes	
timer:SystemTimer	An attribute which produces the SystemTimer.
dStations:List<DStation>	A list attribute which holds all docking stations added to the system.
users:List<UserAccount>	A list attribute which holds all users registered into the system.
rentEntries:List<RentEntry>	A list attribute which holds all Rent Entries added to the system.

Operations	
generateUsage(): usageData	An operation which returns usageData to be used at the Hub Station (use-case 7).
checkStatus(): Status	An operation which is used to display the current capacity of all docking stations.
registerStation()	An operation used to register a new Docking Station within the Hub.

Hub Station Class (associated with Hub Class).

This abstracts the monitor, mouse and keyboard the the operator may use to get reports from the Hub**.**

Attributes.	This class holds no attributes.

Operations.	
generateUsage()	An operation which generates a usage report (use-case 7).
displayUsage(UsageData)	An operation which displays the usage report, created from the usageData argument.

Hub Interface.

Operations.	
getNearby()	An operation which returns the nearbyMap of the Docking Station (use-case 6). This also extends the loan time of the Rent Entry added.
registerUser(persDetails, authCode, key)	An operation which takes persDetails, authCode and key as arguments and returns a User Account, hence storing it into the Hub list of users (use-case 1).
getSummary (ua:UserAccount)	An operation which takes a User Account as an argument and returns its activity summary, using the list f RentEntries. (use-case 4).
addRentEntry()	An operation which adds a new RentEntry to the Hub System (use-case 2).

Docking Station class.

Attributes	
DSId	An integer attribute which Identifies the Docking Station within the Hub System.

dPoints	A list attribute containing of all the docking points within the specific station.
isFull	A boolean attribute which shows whether the Docking Station has reached maximum capacity or not (True if it is at maximum capacity, False if there are empty slots).
isEmpty	A boolean attribute which shows whether there are any available bikes at the specific Docking Station (True if there are no available bikes, False otherwise).
location	A Location attribute which is used to determine the nearby Docking Stations.

Operations	
getNearby(): nearbyMap	An operation which returns the nearbyMap of Docking Stations (use-case 6). This operation also passes the key inserted by the user to the Hub in order to extend the loan time for that RentEntry.
registerUser(personalDetails, authCode, key)	An operation that takes personalDetails, authCode and key as attributes in order to pass them to the Hub (use-case 1).
getSummary(uA: UserAccount)	An operation which takes a UserAccount as an attribute in order to determine its trip summary, using the Hub.(use-case 4).
addRentEntry()	An operation which adds a new rent entry when hiring a bike (use-case 2).

Docking Station Interface.

The DockingStation interface has a getFaultyBikes() operation that the Hub can use to display faulty bikes on the wall screen. We also provide isFull() and isEmpty() operations that the Hub can access. The way the Docking Station uses this methods is by going through the statuses of all the Docking Points, though the actual implementation of the methods is not important at this stage.

Operations.	
getFaulty()	An operation which checks all docking points for a True isFaulty attribute in order to determine which hold faulty bikes.
isFull()	An operation which returns a Bool variable to update the Docking Station status within the Hub.
isEmpty()	An operation which returns a Bool variable to update the Docking Station status within the Hub.

Terminal Class (input/output device associated with Docking Station class).

Attributes	
dStation	A Docking Station attribute which identifies within which Docking Station the terminal is localized.

Operations	
getNearby(): nearbyMap	An operation which returns a nearbyMap of Docking Stations (use-case 6). This operation also passes the key inserted by the user to the Docking Station in order to extend the loan time for that RentEntry.
displayNearby(): nearbyMap	An operation which displays the nearbyMap of Docking Stations (use-case 6).

getSummary(uA: UserAccount)	An operation which takes the UserAccount as an attribute in order to determine its activity summary (use-case 5).
displaySummary(summary)	An operation which takes summary as an attribute in order to display it (use-case 5).
registerUser(personalDetails, authCode, key)	An operation which takes personal Details, authCode and key as attributes in order to register the user within the Hub system (use-case 1).

Rent Entry Class (associated with Hub class).

Attributes	
account	A User Account attribute which holds the personal details of the user who is hiring a bike.
bike	A BikeId attribute which holds the id of the rented bike.
startTime	A time attribute which holds the start time of the hiring.
endTime	A time attribute which holds the end time of the hiring.

Operations	
newEntry(UserAccount, Bike)	An operation which takes two attributes, namely UserAccount and Bike, in order to add a new hiring entry into the system (use-case 2).
updateEndTime(Time)	An operation which takes a Time attribute in order to update the end time of the hiring in case an extension is needed (use-case 3).

User Account Class (associated with Bike class).

Attributes	
details	A Personal Details attribute.
rentHistory	A list attribute which holds the rent entries made by the user.

Operations	This class holds no operations.

Bike Class (associated with Docking Point class).

Attributes	
bike	An BikeId attribute which holds the bike's identification number.
rented	A boolean attribute which returns the status of the bike (False if available, True if rented).

Operations	This class holds no operations.

Card and PIN Reader Class (input device associated with Docking Station class).

Attributes	This class holds no attributes.

Operations	
getCardDetails(): authCode	An operation which checks the card details and returns an authorisation code, provided the card is valid (use-case 1).
displayPrompts()	An operation which displays prompts, such as "Insert Card", "Enter PIN", "Remove Card" or "Valid Card" .

Docking Point Class.

Attributes	
DPid	An integer attribute which holds the unique identification number for the Docking Point.
bikeAvailable	A boolean attribute which holds the availability of a bike at the specific docking point (True if available, False if not available).
isFaulty	A boolean attribute which indicates whether the bike stored at that specific Docking Point is Faulty or not (True if bike is faulty, False if it is not faulty).

Operations	
reportFaulty()	An operation that is used to report a bike which has developed a fault.(used in extensions)
hireBike()	An operation used to hire the bike stored at that specific Docking Point (use-case 2).
insertBike()	An operation used to insert a bike at that specific Docking Point, upon returning it (use-case 3).
returnBike()	An operation when returning a bike (use-case 3).
updateStatus()	An operation used when hiring or returning a bike, in order to update the status of the Docking Point, and hence change its status within the Docking Station and Hub System. When a bike is hired, the bike field is set to null, and when it’s returned it is set to the bikeId of the bike. (use-cases 2 and 3).

Bike Sensor Class (input device associated with Docking Point class).

Attributes	
bikeDetected	A boolean attribute which has a True value if a bike is detected at the docking point and False otherwise.

Operations	
detectBike()	An operation which uses the sensor in order to determine whether a bike is at the Docking Point or not (use-case 3).

Lock Actuator Class (output device associated with Docking Point class).

Attributes	
locked	A boolean attribute that is True if the bike is locked and False if the bike is unlocked.

Operations	
lock()	An operation used to lock the bike into the Docking Point (use-case 3).
unlock()	An operation used to unlock the bike into the Docking Point (use-case 2).

Smart Key Reader Class (input device associated with Docking Point class).

Attributes	This class holds no attributes.

Operations	
readKey(k: Key)	An operation which reads the contactless electronic key in order to unlock the bike at the Docking Point (use-case 2).

Fault Button Class (input device associated with Docking Point class).

Attributes	This class holds no attributes.

Operations	
press()	An operation which enables the user to press the fault button to report a defective bike.

Light Class (output device associated with Docking Point class).

Attributes	
state	A Colour attribute which determines the light's colour: Red for faulty bikes or unsuccessful attempts to unlock a bike (e.g. using a key which is not valid) and Green for successfully locking or unlocking a bike.

Operations	
turnOn(Colour)	An operation that turns on the light of the Colour taken as an attribute.
turnOff()	An operation that turns off a previously turned on light.
flash(Colour)	An operation which turns on and off the given Colour, such as Green for successfully locking or unlocking a bike at the Docking Point (use-cases 2 and 3).

OK Light Class (output device associated with Light class).

Attributes	
state	A Green colour attribute.

Operations	
flash(Green)	An operation which turns on an off the Green colour (use-cases 2 and 3).

Fault Light Class (output device associated with Light class).

Attributes	
state	A Red colour attribute.

Operations	
turnOn(Red)	An operation which turns on the Red colour in order to report a fault.
turnOff()	An operation which turns off the Red colour at that particular docking point.

Colour Class (utility class).

Attributes	
Red	A String attribute.
Green	A String attribute.

Operations	
set(Red)	An operations which sets the colour to Red and therefore returns Red.
set(Green)	An operations which sets the colour to Green and therefore returns Green.

Key Class (utility class).

Attributes	
id	A KeyId attribute which identifies the dispensed key.

Operations	This class holds no operations.

Smart Key Dispenser Class (associated with Terminal class).

Attributes	This class hold no attributes.

Operations	
dispenseKey(): Key	An operation which dispenses an electronic contactless key used to unlock bikes from their Docking Points (use-case 2).

Wall Display Class (associated with Hub class).

Attributes	This class holds no attributes.

Operations	
showStatus(): Status	An operation which displays the status of all Docking Stations on the Wall Display.

System Timer Class (associated with Hub class).

Attributes	This class holds no attributes.

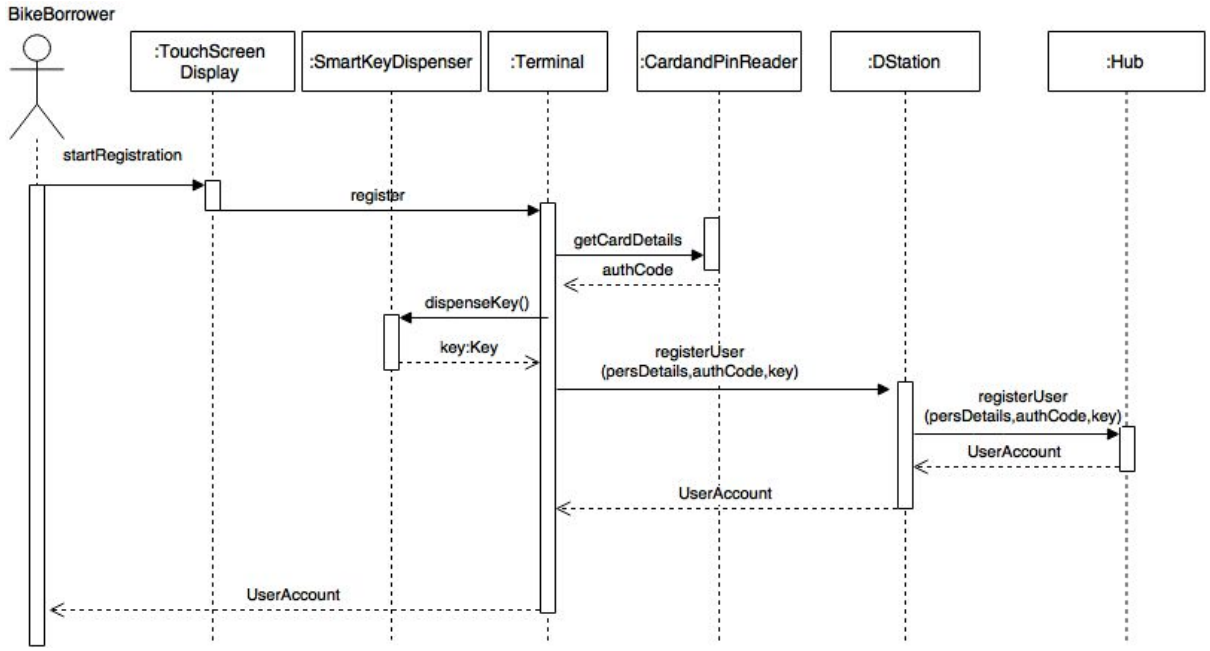
Operations	
charge()	An operation which is used to charge all registered users every midnight for their hired bikes throughout the past day (use-case 5).

Bank Server Interface.

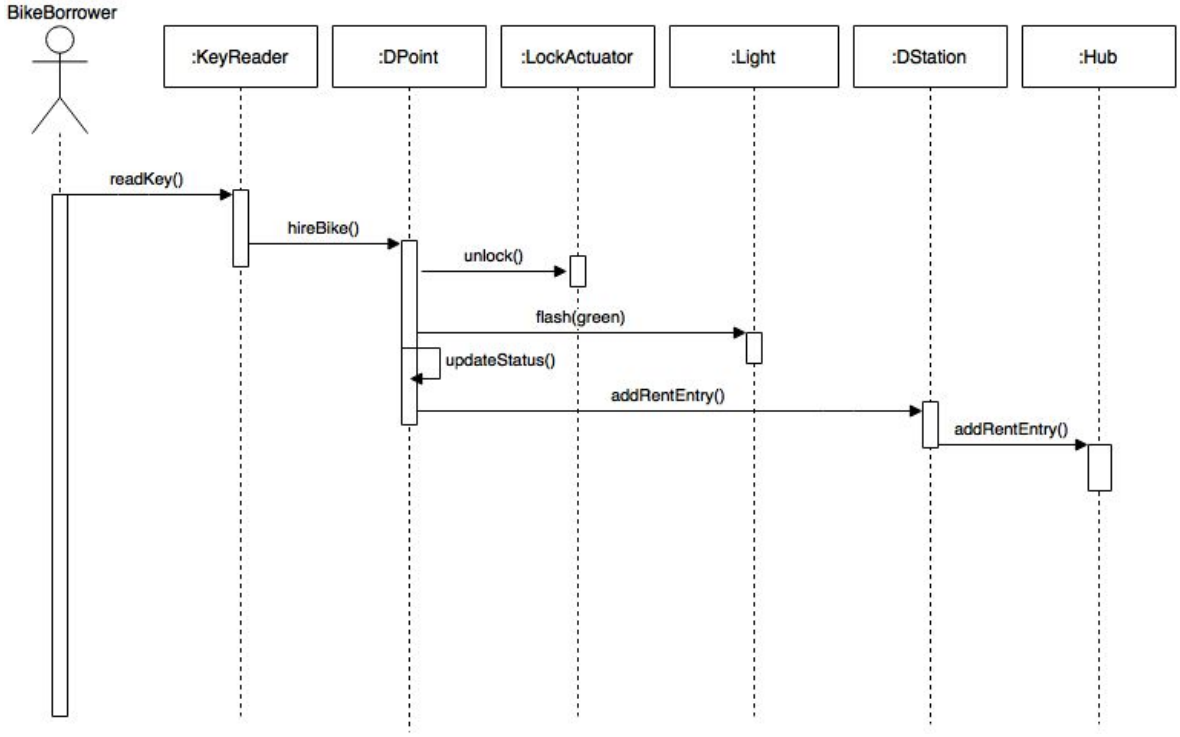
Operations.	
chargeUser()	An operation which charges all users at midnight.

2.4. Dynamic Models (UML Sequence Diagrams)

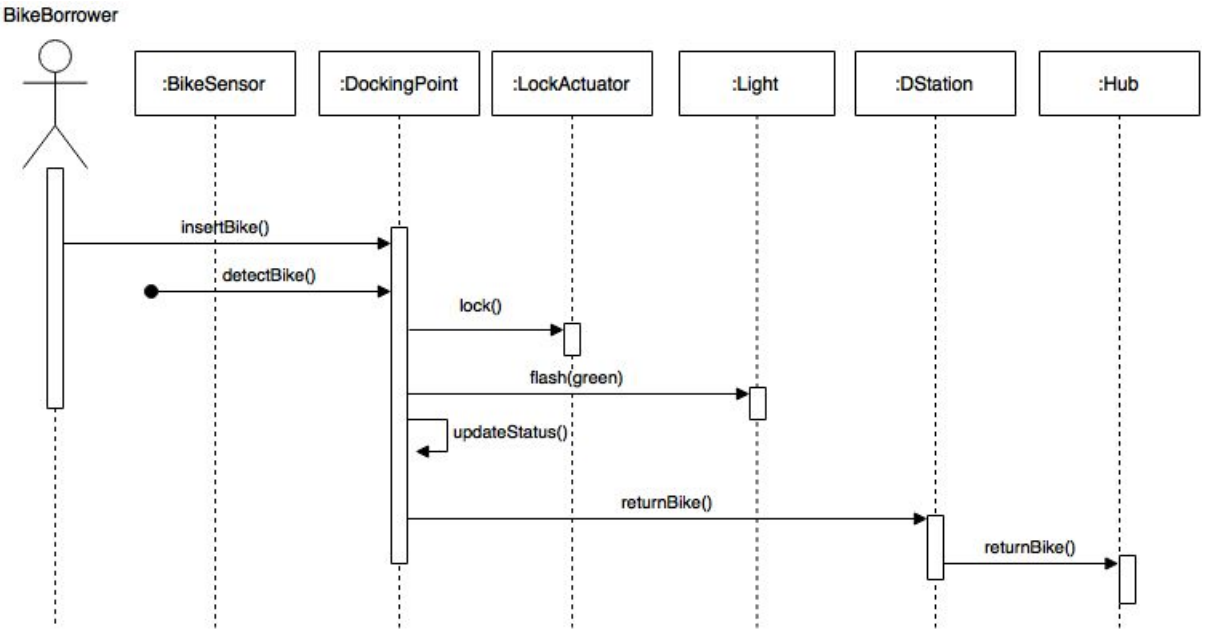
- Sequence diagram for Registering a user.



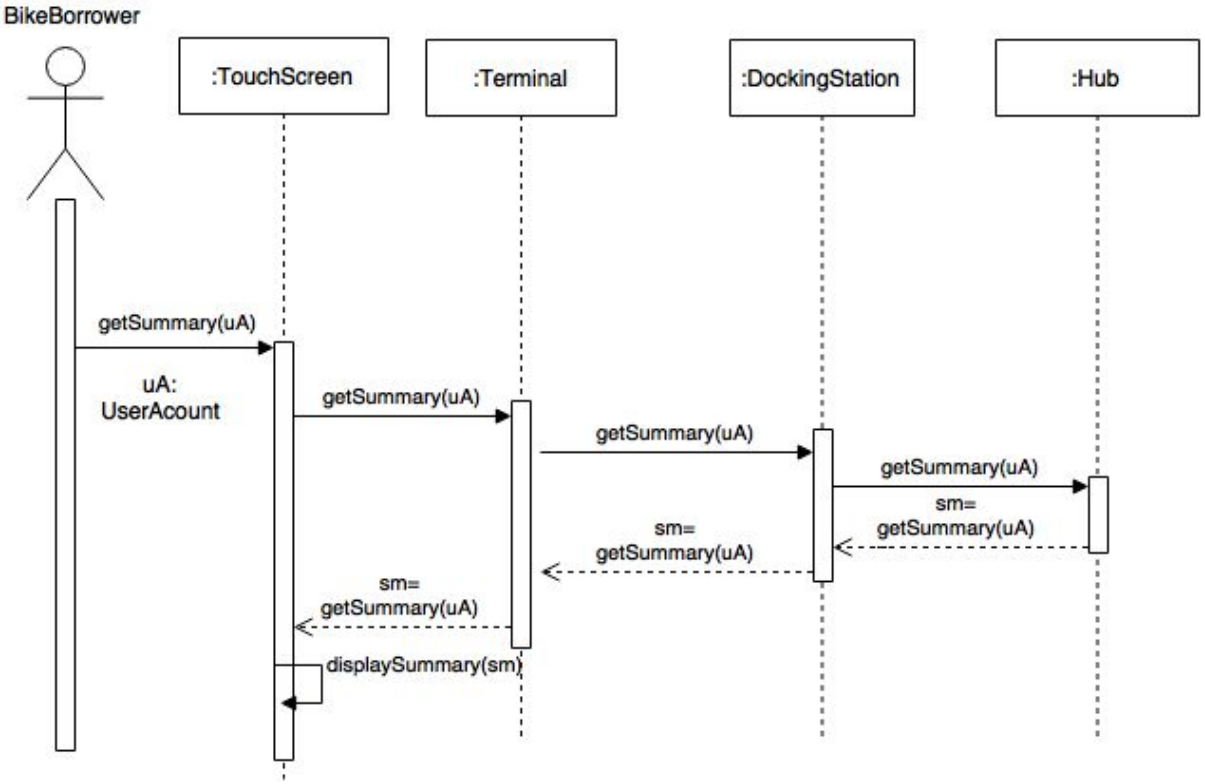
- Sequence diagram for hiring a bike.



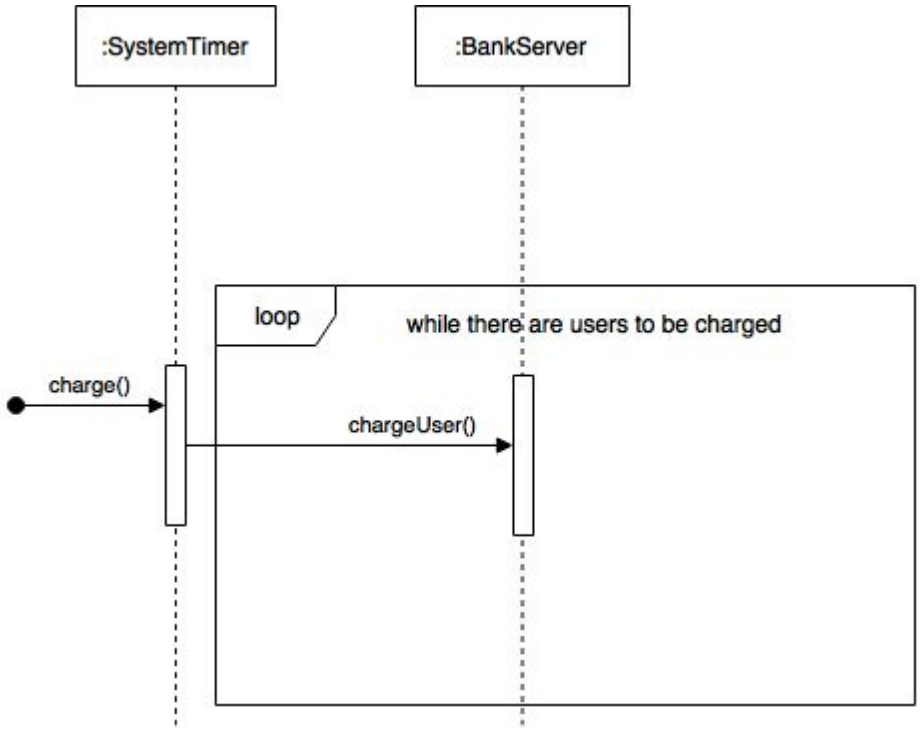
- Sequence diagram for returning a bike.



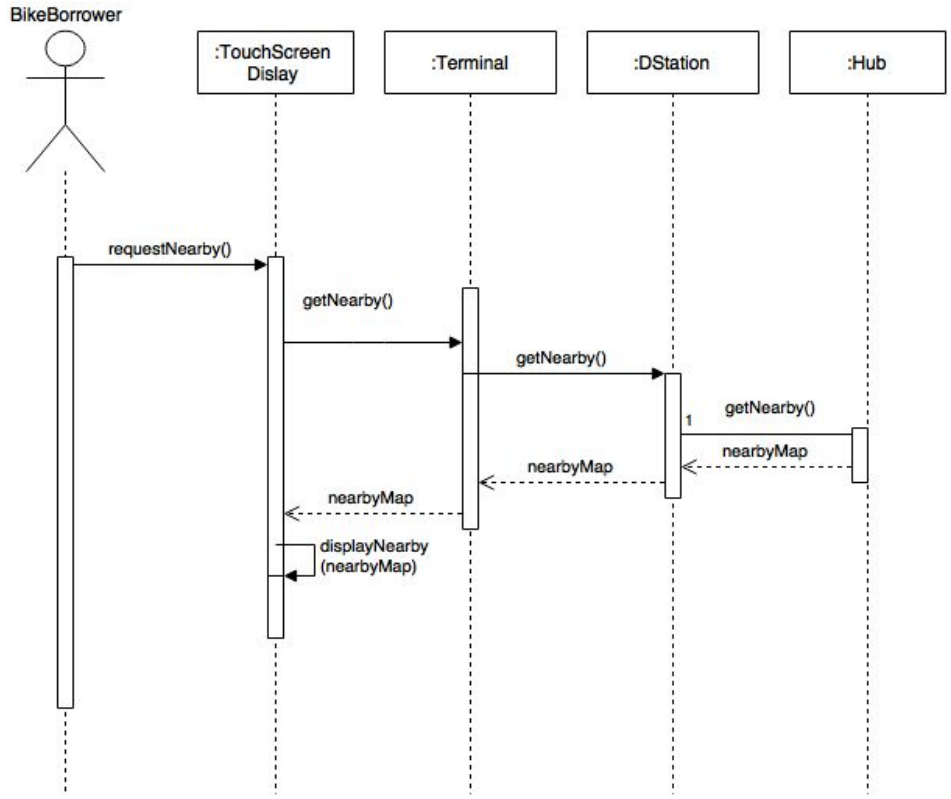
- Sequence diagram for generating the activity summary for the user.



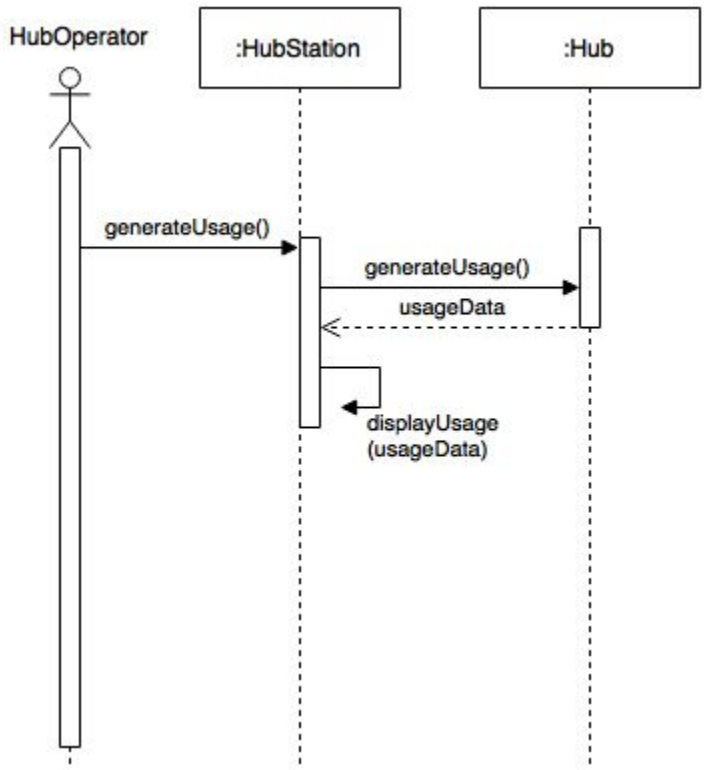
- Sequence diagram for charging users.



- Sequence diagram for showing nearby docking stations with free docking points.



- Sequence diagram for generating a report.



2.5. Extension Scenarios

Extension to Returning a bike: Bike is faulty.
Our design accounts for this situation by providing a fault button that the user can press after inserting the bike into the docking point. The fault button updates status of the docking point.

Extension to Returning a bike: All docking points are occupied
User can select a list of nearby docking stations. He's rentEntry is updated to grant 15 additional minutes, by the getNearby() method..

Extension to Hiring bike: No bikes available
Our design allows user to request a list of nearby docking stations, generated by the hub.

Extension to Registering: Card not valid
On registration, system checks first the validity of the credit card. The verification is done by the bank server, returning an authorisation code or an error.

2.6. Changes in requirements and use-cases

- We have added the missing Inputs/Outputs, more specifically Smart Key Dispenser and Smart Key Reader, hence changing the classes we previously used and sorting the mismatches in the Requirements list.
- We have remedied the absence of Secondary Actors from our first Coursework by dividing our Use Cases into smaller Use Cases and hence eliminating the confusions created (for example we have removed the Registering part from our Hiring Bike Use Case scenario).

2.7. Summary of the use-cases

1. Register user: User inserts details and credit card; an account is created and he is issued a key.
2. Hire bike: User inserts key to unlock a bike.
3. Return bike: User returns bike to docking station.
4. Generate activity summary for user: User request an activity summary for his account.
5. Charge user: System Timer charges all users at midnight.
6. Show nearby docking stations with free points: User requests a map with nearby docking stations and the terminal displays is.
7. Generate report: Hub operator requests an usage report of all docking stations.