Evidence Gathering Document for SQA Level 8 Professional Developer Award.

This document is designed for you to present your screenshots and diagrams relevant to the PDA and to also give a short description of what you are showing to clarify understanding for the assessor.

Each point that required details the Assessment Criteria (What you have to show) along with a brief description of the kind of things you should be showing.

Please fill in each point with screenshot or diagram and description of what you are showing.

Week 2

Unit	Ref	Evidence
I&T	I.T.5	Demonstrate the use of an array in a program. Take screenshots of: *An array in a program *A function that uses the array *The result of the function running
		Description: Top: An array of Peanuts characters names, with a function to add a character to the end of the array. **Bottom:** The array that is produced as a result of the function to add the character.

```ruby
# Array

peanuts = ["Charlie Brown", "Linus", "Lucy", "Woodstock",
"Snoopy"]

def add_character(array, name)
  array << name
end

add_character(peanuts, "Schroeder")
p peanuts
```

```
➜  extra ruby extra.rb
["Charlie Brown", "Linus", "Lucy", "Woodstock", "Snoopy", "Schroeder"]
```

Unit	Ref	Evidence
I&T	I.T.6	Demonstrate the use of a hash in a program. Take screenshots of: *A hash in a program *A function that uses the hash *The result of the function running
		Description: Top: An example of a hash of Dr. Seuss books, which contain the titles of the books and the number of pages. The function that uses the hash loops through the hash and adds each books number of pages to figure out how many pages in total will be read. **Bottom:** Is the output result of the function running which produces a string and the sum of all the pages.

```ruby
# Hash
books = { title: "Oh, the Places You'll Go!", author: "Dr. Seuss", pages: 48 }

def get_title(books)
  return "You have read " + books[:title] + " by " + books[:author] + "."
end

p get_title(books)
```

```
➜  extra ruby extra.rb
"You have read Oh, the Places You'll Go! by Dr. Seuss."
```

Week 3

Unit	Ref	Evidence
I&T	I.T.3	Demonstrate searching data in a program. Take screenshots of: *Function that searches data *The result of the function running
		Description: The function below is an album function, to search the data using the album id and return the artist information. **Top**, is the function; **bottom**, is the result of the function being run for album 1.

```
def artist()
  sql = "SELECT * FROM artists WHERE id = $1"
  values = [@id]
  results = SqlRunner.run(sql, values)
  return Artist.new(results[0])
end
```

```
[1] pry(main)> album1.artist
=> #<Artist:0x007fd5c8367ab8 @id=1, @name="Madonna">
```

Unit	Ref	Evidence
I&T	I.T.4	Demonstrate sorting data in a program. Take screenshots of: *Function that sorts data *The result of the function running
		Description: The function displayed below (self.all_order), **left top**, selects all the albums in the music collections and then sorts them in descending order by title. The result of the function being run, **right**, shows the details of the 4 albums in the music collection, displaying them in descending alphabetical order (Z-A). **Bottom left** image shows the same result but in a PSQL table.

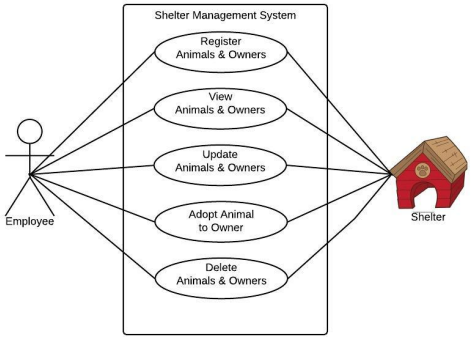
```
def self.all_order()
  sql = "SELECT * FROM albums ORDER BY title
  DESC"
  albums_order = SqlRunner.run(sql)
  return albums_order.map { |album|
  Album.new(album) }
end
```

```
music_collection=# SELECT * FROM albums ORDER BY title DESC;
 id |    title     | genre | artist_id
----+--------------+-------+-----------
  3 | True Blue    | Pop   |         1
  1 | Ray of Light | Pop   |         1
  4 | Music        | Dance |         1
  2 | Blue Hawaii  | Pop   |         2
(4 rows)
```

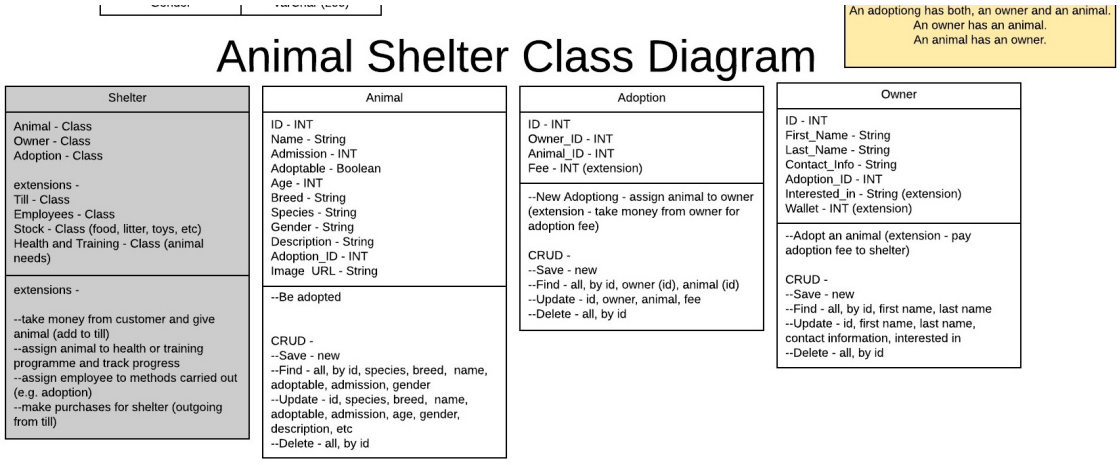
```
[3] pry(main)> Album.all_order
=> [#<Album:0x007f9caa0a4800
  @artist_id=1,
  @genre="Pop",
  @id=3,
  @title="True Blue">,
 #<Album:0x007f9caa09fd50
  @artist_id=1,
  @genre="Pop",
  @id=1,
  @title="Ray of Light">,
 #<Album:0x007f9caa09f710
  @artist_id=1,
  @genre="Dance",
  @id=4,
  @title="Music">,
 #<Album:0x007f9caa09f3a0
  @artist_id=2,
  @genre="Pop",
  @id=2,
  @title="Blue Hawaii">]
[4] pry(main)>
```

Week 5 and 6

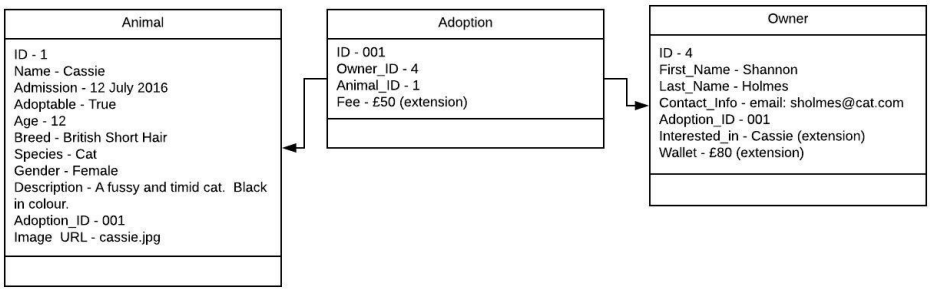
Unit	Ref	Evidence
A&D	A.D.1	A Use Case Diagram
		Description: The use case diagram below shows the Animal Shelter system and how the employee will interact with it.



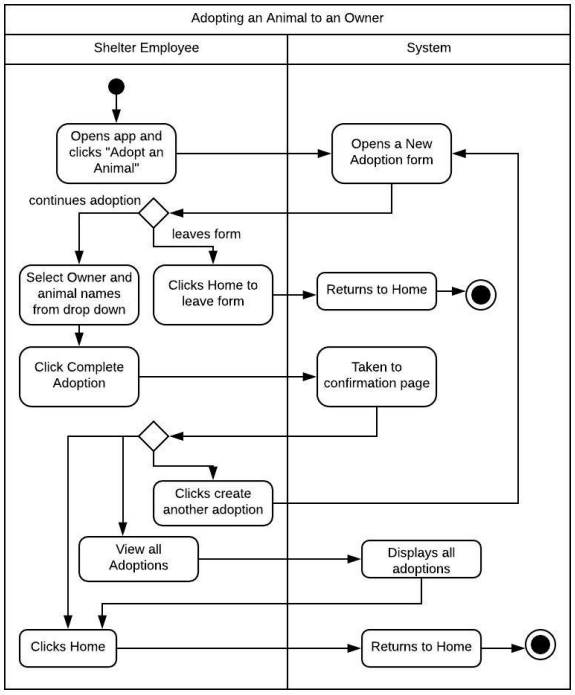
Unit	Ref	Evidence
A&D	A.D.2	A Class Diagram
		Description: The class diagram below shows the 3 main classes: Animal, Adoption, and Owner. It contains all the information required for the MVP, as well as identified extensions relating to classes and their methods.



Unit	Ref	Evidence
A&D	A.D.3	An Object Diagram
		Description: An example object diagram providing examples of each class - Animal, Adoption and Owner. It illustrates that the adoptions class relies on information from the other 2 classes.



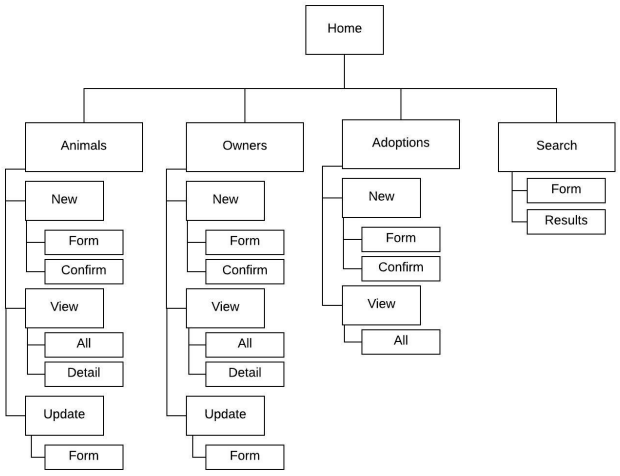
Unit	Ref	Evidence
A&D	A.D.4	An Activity Diagram
		Description: The activity diagram shows the process of creating and saving a new adoption in the system, outlining all the choices the employee has to make and how the system responds.



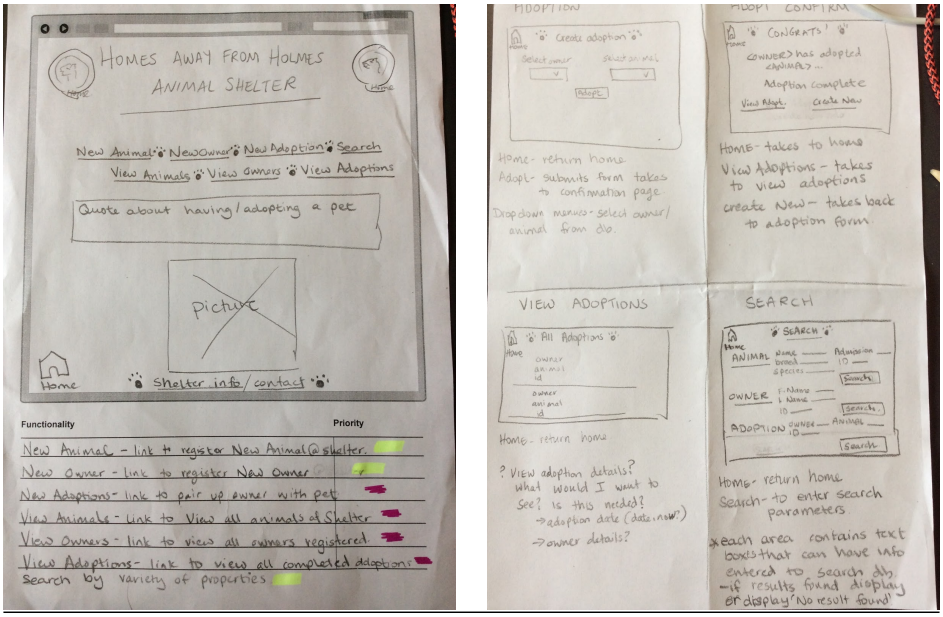
Unit	Ref	Evidence
A&D	A.D.6	Produce an Implementations Constraints plan detailing the following factors: *Hardware and software platforms *Performance requirements *Persistent storage and transactions *Usability *Budgets *Time
		Description: This Implementation Constraints Plan is for the group JavaScript Project. It identifies the limitations and considerations being made that will constrain the product and how it might affect the development.

Implementation Constraints Plan		
Topics	**Possible Effect of Constraint on Product**	**Solution**
Hardware and software platforms	• Not setup to use on mobile devices • Currently only available locally	• Currently not a consideration as first need to have a system in place that carries out basic tasks then can look into updating layout to be responsive to devices if required. • Once satisfactory can be deployed on web
Performance requirements	• Needs to be simple	• Ensure that true and false booleans say yes/ available OR no/not available • clear links/button labels - e.g. not submit but complete adoption, etc.
Persistent storage and transactions	• No place to save updated pictures when uploaded	• While pictures are nice they are not a requirement and do not necessarily need to be part of the form; later date can link to a folder on computer or database where uploaded photos can be stored
Usability	• Only currently usable locally on single computer, data is not available unless at that terminal	• Would need to have a database stored online or computer terminals linked to access information
Budgets		• Currently not an issue but will depend on further needs of the client
Time		• Initial timescale allows for MVP to be created

Unit	Ref	Evidence
P	P.5	User Site Map
		Description: The sitemap, below, shows the various pages that the employee can interact with on the Animal Shelter system, including extension sections (e.g. search).



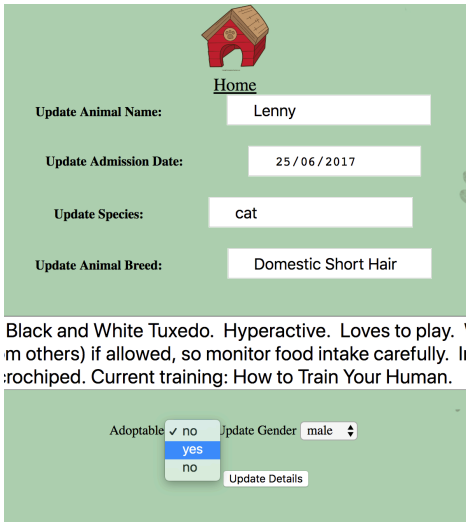
Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams
		Description: The wireframes, below left, show the Animal Shelter main home page with its functionality; below right, shows the 3 main views of the adoption section - the form, confirmation of form submitted, and viewing all adoptions (also included is the view for the extension search page).

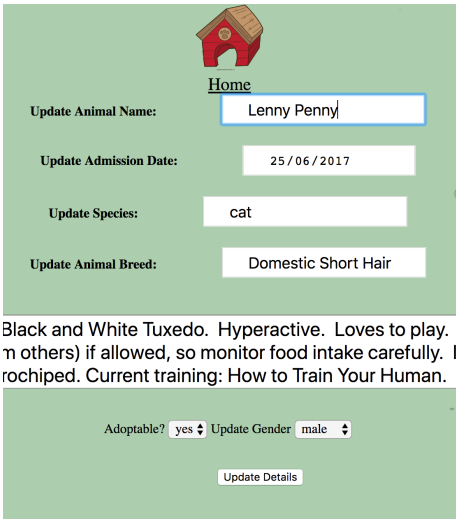


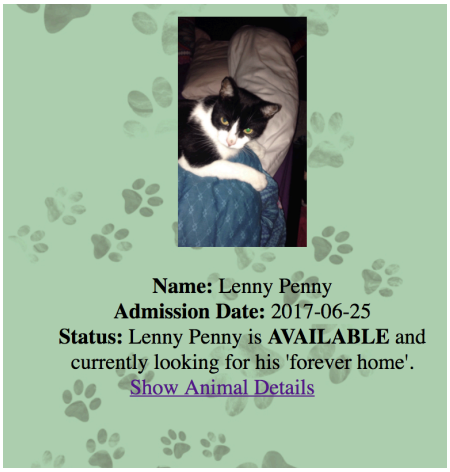
Unit	Ref	Evidence
P	P.10	Example of Pseudocode used for a method
		Description: This is an example of pseudocode that helped to explain the logic behind the process of adopting an animal. It is incomplete as it produces an adoption, however it then does not then remove the animal from the available adoption list which was discovered upon submission was happening. Further work needs to be done to fix this issue, but the pseudocode was successful in making the adoption feature itself work.

```
Pseudocode -
it ('should return class information listing all owners and their adopted
animals', functions() {
  //for each adoption it will need an animal and an owner,
  //an example of an owner to test,
  //an example of an animal to test,
  //an example of an adoption to test,
    //SQL - " select first and last name from the owners table - using ids"
    //"select name from the animals table - using ids"
    //make connection through adooptions table - using owner_id and animal_id.
    //run the SqlRunner method
    // return result including owners full name and name of animal
```

Unit	Ref	Evidence
P	P.13	Show user input being processed according to design requirements. Take a screenshot of: * The user inputting something into your program * The user input being saved or used in some way
		Description: The user is inputting updated information into the programme, **left** and **middle** images, which is changing the animals name and changing whether he is adoptable. The image on the r**ight** shows how the input is saved and used to display the correct information about the animal, so that owners would know that the pet is available.



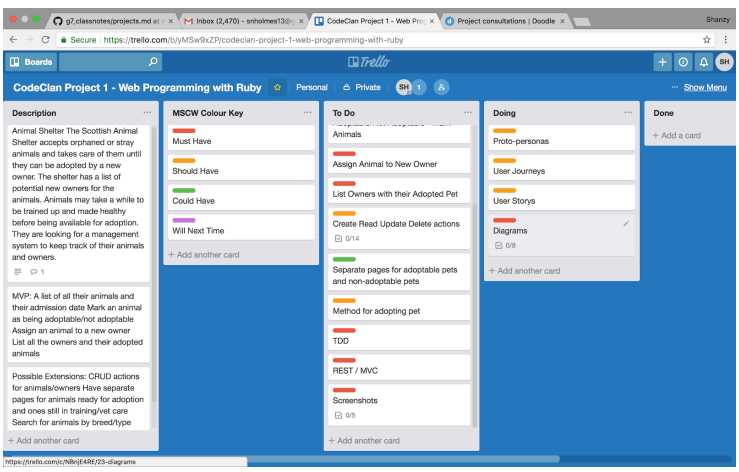


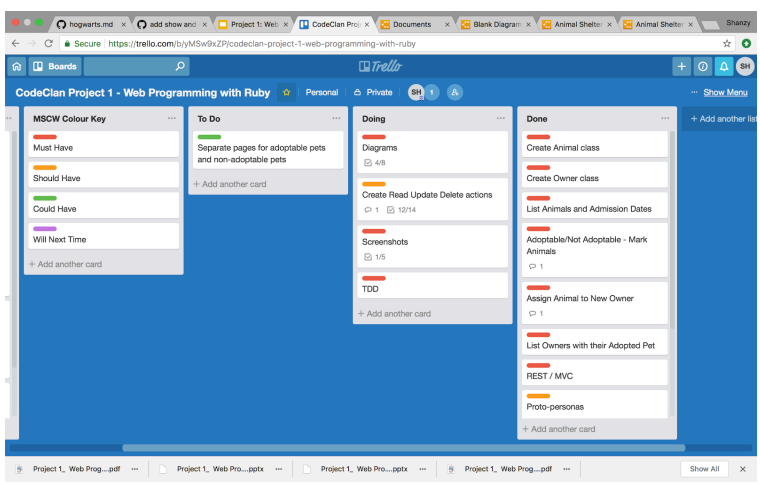


Unit	Ref	Evidence
P	P.14	Show an interaction with data persistence. Take a screenshot of: * Data being inputted into your program * Confirmation of the data being saved
		Description: Below **left**, shows the data, owner and animal names, being input into the programme by the form which sets up a new adoption. **Middle** shows the confirmation of the data being saved and provides a message to show what data was entered in the form and congratulating them on a successful adoption. **Right** shows that the new adoption has been added to the bottom of the all adoptions list.

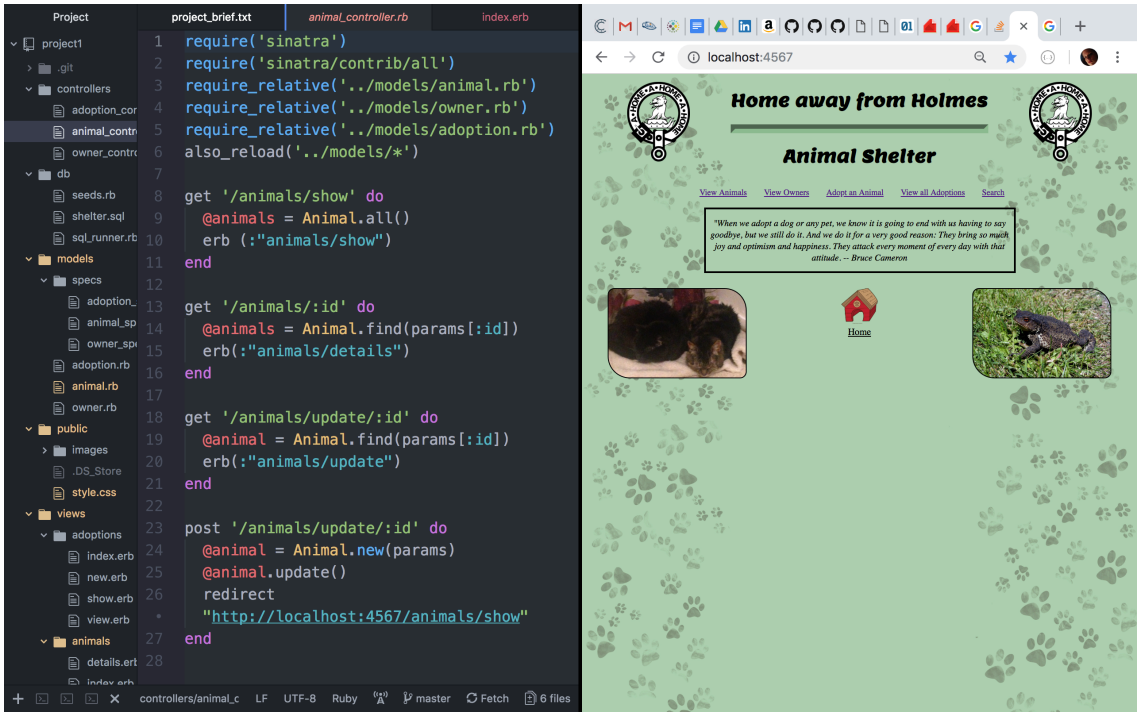
Unit	Ref	Evidence
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: * The user requesting information or an action to be performed * The user request being processed correctly and demonstrated in the program
		Description: Below **left**, demonstrates that a user can request all adoption info by clicking on the link “View all Adoptions”, and below **right** demonstrates the output of their request by opening up a page which shows all adoptions.

Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.
		Description: Below shows the MOSCOW Trello board for the project, **left** is at the start with lots ‘To Do’ and **right** is at the end with all the main Must Haves for the function of the programme in the ‘Done’ section.





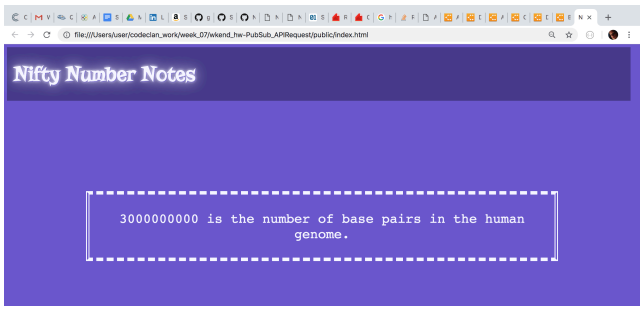
Unit	Ref	Evidence
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.
		Description: Below is a screenshot of a project where I worked alone, showing both the code and the page. It can be found at: https://github.com/ShanzyHolm/Project1-Animal_Shelter-Web_Programming_with_Ruby



Week 7

Unit	Ref	Evidence
P	P.16	Show an API being used within your program. Take a screenshot of: * The code that uses or implements the API * The API being used by the program whilst running
		Description: Below **left** is the request function that uses the API and **bottom right** shows the new request being made to get the data from the API and **top right** shows the result displayed of the request made to the API.

```
extensions_cod...  instructions.txt  index.html  number.js  request.js  number_view.js  select_number_...
1  const Request = function(url) {
2    this.url = url;
3  }
4
5  Request.prototype.get = function(onComplete) {
6    const xhr = new XMLHttpRequest();
7
8    xhr.addEventListener('load', () => {
9      if (xhr.status !== 200) {
10       return;
11     };
12
13     const responseText = xhr.responseText;
14     // const data = JSON.parse(responseText);
15     // Does not need to be parsed as API is sending JSON format already
16     const data = responseText;
17     onComplete(data);
18     // console.log(data);
19   });
20
21   xhr.open("GET", this.url);
22   xhr.setRequestHeader('Accept', 'application/json');
23   xhr.send();
24 };
25
26 module.exports = Request;
27
```




```
9  Number.prototype.getData = function () {
10   const request = new Request('http://numbersapi.com/random/trivia');
11   request.get((data) => {
12     // console.log(data);
13     // console.log(event);
14     this.text = data;
15     // console.log(this.text);
16     PubSub.publish('Number:number-loaded', this.text);
17   })
18 }
```

Unit	Ref	Evidence
P	P.18	Demonstrate testing in your program. Take screenshots of: * Example of test code * The test code failing to pass * Example of the test code once errors have been corrected * The test code passing
		Description: Below are examples of a JavaScript code failing to pass. On the **left** is the test code and the unwritten code that is needed to get the test to pass. On the **right** is the completed code that corrects the error and has a passing test.

```
1  var Maths = require('./main.js')
2
3  var add = Maths.add
4
5  var expect = require('chai').expect
6
7  describe("add", function(){
8    it("2 and 2 to make 4", function(){
9      expect(add(2, 2)).equal(4)
10   })
11 })
```

```
1  const add = function(x, y){
2
3  }
4
5  module.exports = {add: add};
6
```

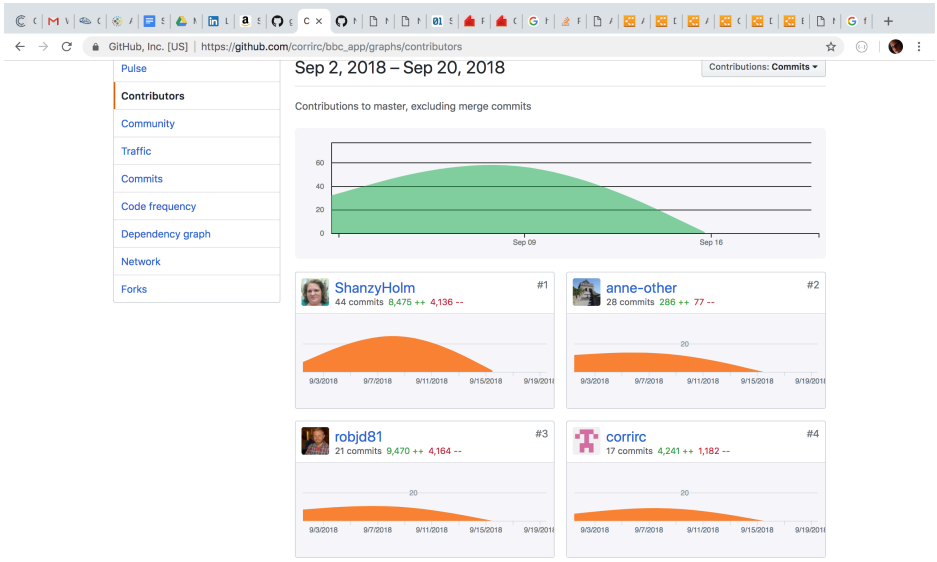
```
add
  1) 2 and 2 to make 4


0 passing (10ms)
1 failing

1) add
     2 and 2 to make 4:
   AssertionError: expected undefined to equal 4
    at Context.<anonymous> (main_spec.js:10:23)


npm ERR! Test failed.  See above for more details.
```

```
1  const add = function(x, y){
2    return x + y
3  }
4
5  module.exports = {add: add};
6
7
8
```

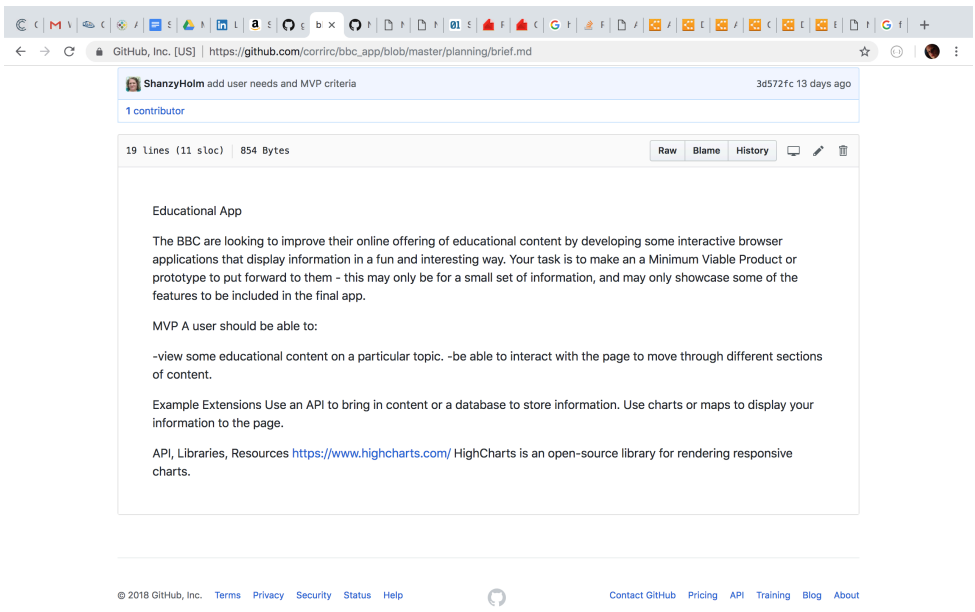
```
→  mocha_playground npm test main_spec.js

> mocha_playground@1.0.0 test /Users/user/co
> mocha "main_spec.js"



  add
    ✓ 2 and 2 to make 4


  1 passing (7ms)

→  mocha_playground
```

Week 9

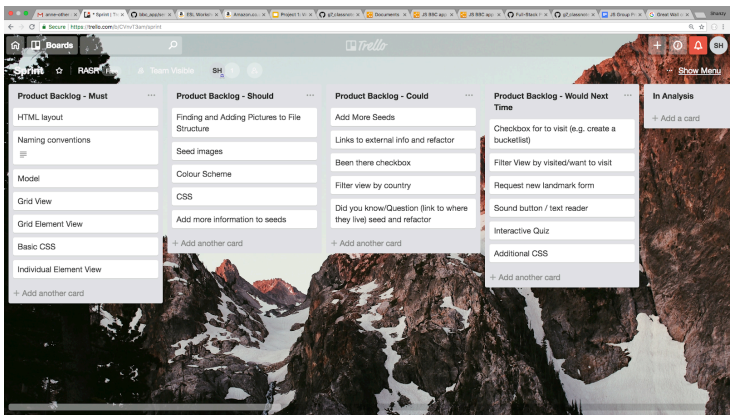
Unit	Ref	Evidence
P	P.1	Take a screenshot of the contributor's page on Github from your group project to show the team you worked with.
		Description: https://github.com/corrirc/bbc_app. This is the contributors page from the group JavaScript project.

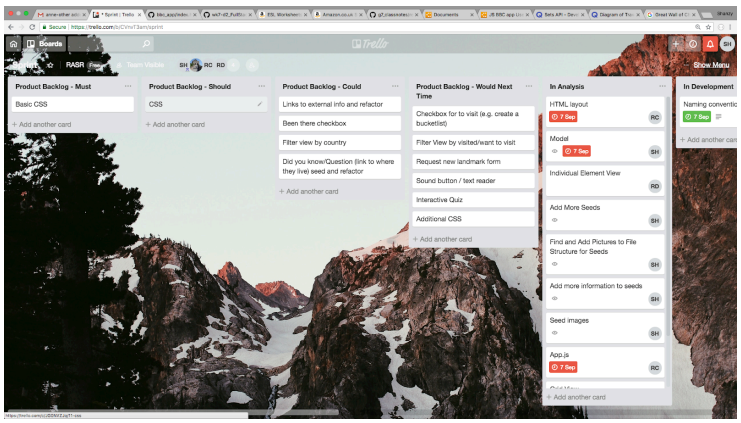


Unit	Ref	Evidence
P	P.2	Take a screenshot of the project brief from your group project.
		Description: This is the project brief from the JavaScript group project for the educational app, showing the MVP that we had to further define and develop as we began planning out Landmark App.



Unit	Ref	Evidence
P	P.3	Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.
		Description: On the **left** is the Trello KanBan board showing our project backlog with the criteria we felt we much have, should have, could have and would not have this time. The version of the **right** shows that must and should no longer have lists of criteria attached to them and even the could list has less.





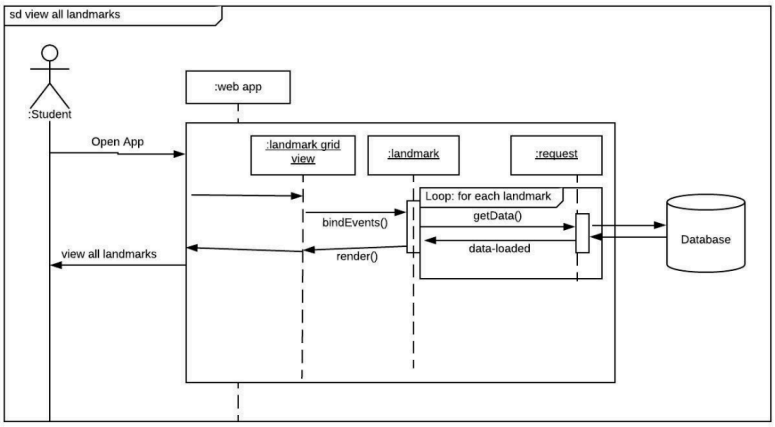
Unit	Ref	Evidence
P	P.4	Write an acceptance criteria and test plan.
		Description: The acceptance criteria and test plan below are both for the group JavaScript project, and displays the 2 main MVP criteria and 3 of the extensions. By the time we presented each criteria had produced the expected outcome and passed.

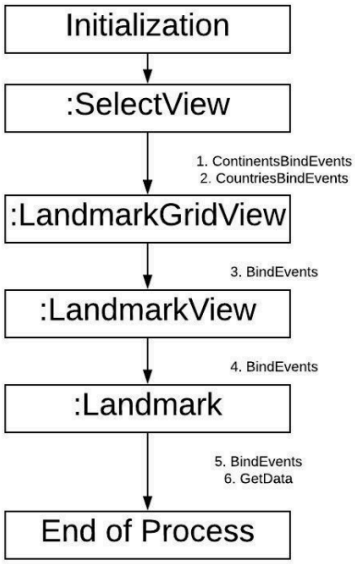
Acceptance Criteria

Acceptance Criteria	Expected Result/Output	Pass / Fail
Student is able to access a list of all landmarks	A list of all landmarks is displayed when URL is accessed	**Pass**
A student can click on a landmark to display all of its information	When student clicks, a view is populated with all the landmark information	**Pass**
Extensions		
A student can click on a link inside the landmark view to access external webpage about the landmark	When student clicks the "Get More Information" button, they are taken to an external URL (if landmark has its own page, otherwise a National Geographic or Lonely planet Link)	**Pass**
A student can mark a landmark as visited	Student can ~~check a box~~ toggle a switch to mark a landmark as one they have visited	**Pass**
Student can view a map of landmark location	When student clicks, a map is included in the landmark view as part of the information	**Pass**

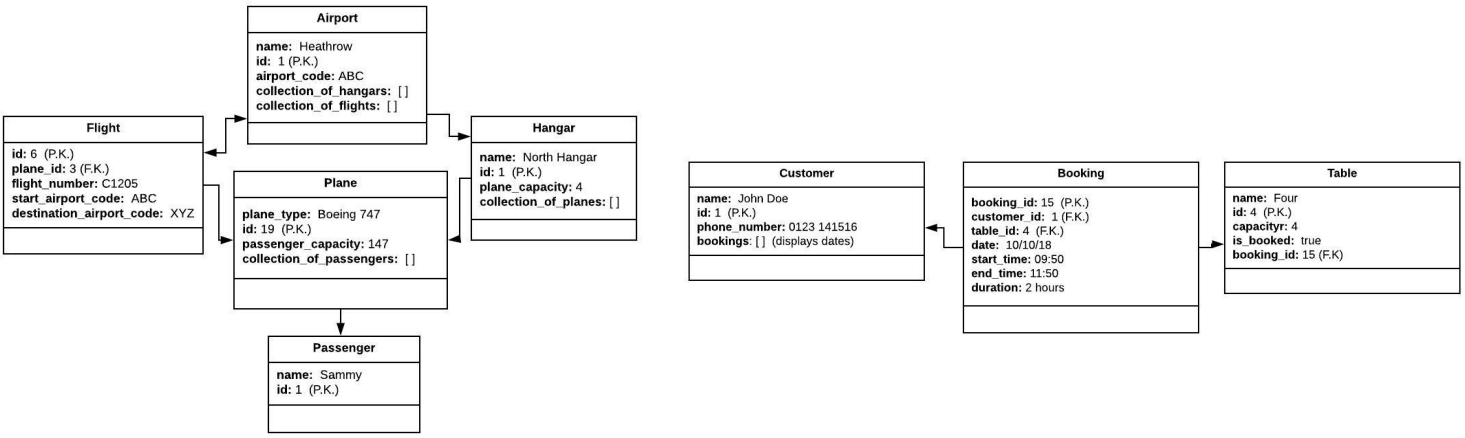
Unit	Ref	Evidence
P	P.7	Produce two system interaction diagrams (sequence and/or collaboration diagrams).
		Description: The system interaction diagrams below are both for the group JavaScript project. The first is a sequence diagram showing the system process that happens when a user wants to view all the landmarks on a page. The collaboration diagram shows how the views and bind events collaborate from start to finish to display a landmark.

Collaboration diagram of a BBC educational landmark app





Unit	Ref	Evidence
P	P.8	Produce two object diagrams.
		Description: Left diagram shows the object diagrams for an airport. Each airport has a collection of flights and hangars, each hangar has a collection of planes, each flight has a start and end airport as well as a plane, and each plane has a collection of passengers. **Right** diagram shows the customer, booking and table object diagrams are from the group Java project and show what information both the customer, table and bookings would have as well as how the 3 objects would connect, the customer having a list of bookings and the booking having a customer.



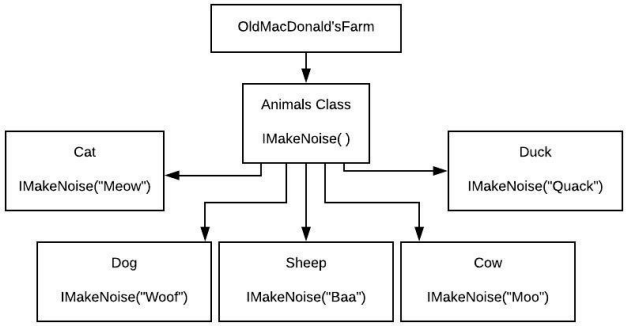
Unit	Ref	Evidence
P	P.17	Produce a bug tracking report
		Description: This is the Bug Tracking Report for the group JavaScript project. The third one down we didn't think was going to work as we weren't sure if it would be able to pass the URL from the seeds and make it a clickable link that the user could interact with, however of all the issues we encountered this turned out not to cause us any issues - it was just down to how we approached entering the URL in the seeds and how we coded it to display the link. The other bugs turned out to be a lot more complicated to figure out and solving them took a bit longer.

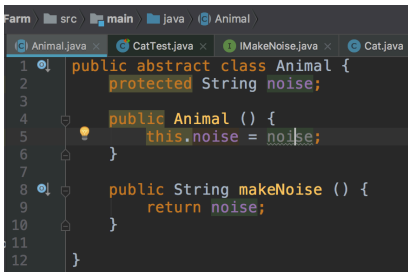
Bug Tracking Report

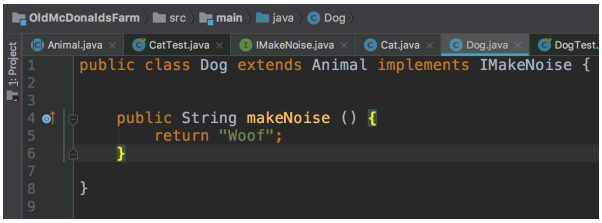
Issue		Solution	Pass/ Fail
Database can load seeds	Fail	Fix syntax errors in seeds	Pass
Student can click on a landmark and page will scroll down to the information.	Fail	Fix placement of scroll method within the code to allow content to load fully before scroll occurs.	Pass
Student can click on link within landmark view to be taken to an external website.			Pass
Toggle switch is displayed on the grid item view for student to interact with	Fail	Added missing line of code in CSS file	Pass
Toggle switch will update the database with changes (true or false) as to whether student has been to the landmark	Fail	Fixed code so that it was updating a single parameter within the database rather than the whole object.	Pass
Landmark view will load map to show location of the selected landmark While it worked for some landmarks, it did not work for all (as bringing back local places "Sphinx Medical" or some not even found)	Fail	Add additional parameters to the method to get name, location and continent of landmark (Future: Add lat/lng to seeds?)	Pass

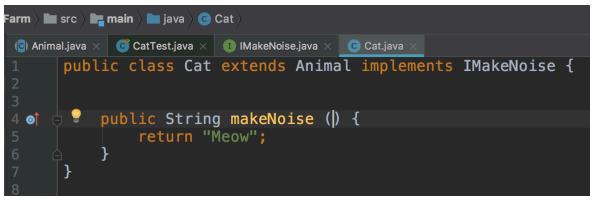
Week 12

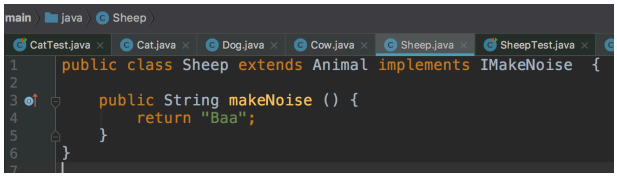
Unit	Ref	Evidence
I&T	I.T.7	The use of Polymorphism in a program and what it is doing.
		Description: **Below** is an example of Polymorphism, at Old McDonalds Farm each of the animals have the action of making noise, though what they do is not the same. So one makeNoise method is written and each animal is identified as IMakeNoise by implementing that interface and extends Animal where the makeNoise method is defined. Each animal though returns a different noise when makeNoise is called. It allows one method to be written but for it to be implemented in different ways or for it to produce a different result.

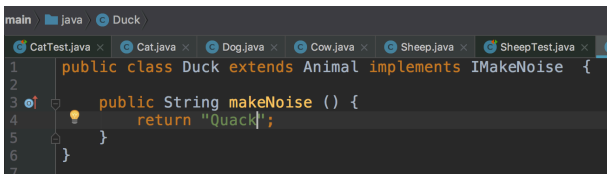


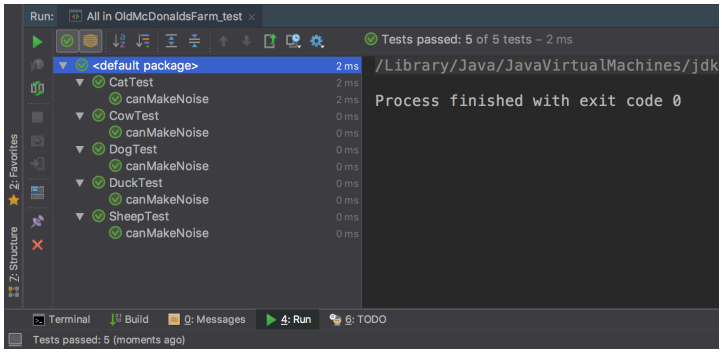




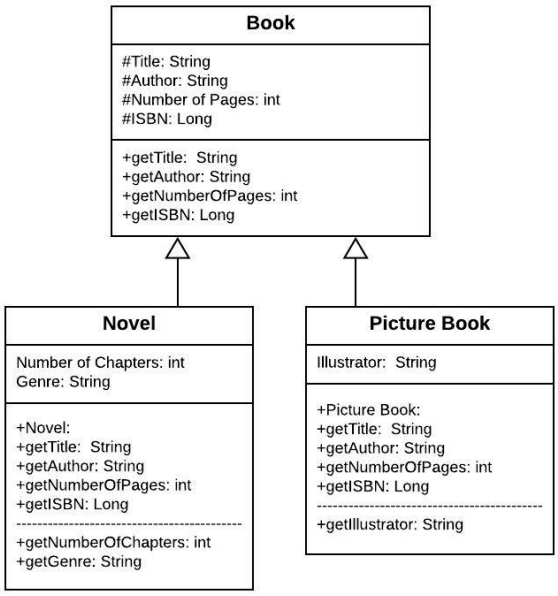




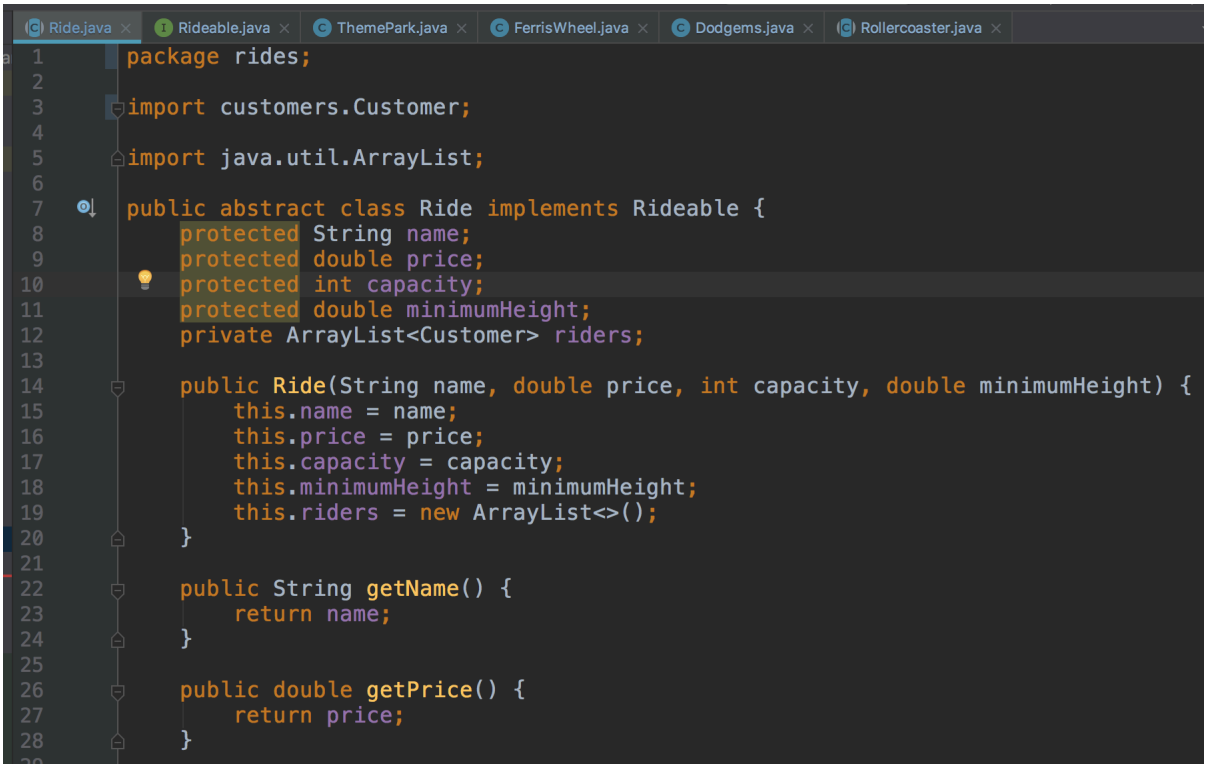




Unit	Ref	Evidence
A&D	A.D.5	An Inheritance Diagram
		Description: This inheritance diagram shows that all books have a Title, Author, Number of Pages, and ISBN. It then shows how both Novels and Picture Books inherit those properties and methods, but at the same time also have their own properties and methods specific to them, such as a Novel has chapters whereas Picture Books do not.



Unit	Ref	Evidence
I&T	I.T.1	The use of Encapsulation in a program and what it is doing.
		Description: This example shows encapsulation as it has properties that are only available and viewable within he Ride class itself. It has methods for publicly accessing the information, such as getName and getPrice. It has a private array of riders that is only accessible in the Ride class, and it has protected information which can be accessed and used by subclasses, including the name, price, capacity and minimum height. In this example the name and price of each ride can be accessed throughout the theme park if they call the public methods. However the name, price, capacity and minimum height will vary and need to be modified by subclasses of Ride so that it is specific to them, they all must contain this information. Each ride will also have riders however only the Ride class can access the original array, the subclasses will create duplicates or new Arrays to contain their individual information.



Unit	Ref	Evidence
I&T	I.T.2	Take a screenshot of the use of Inheritance in a program. Take screenshots of: *A Class *A Class that inherits from the previous class *An Object in the inherited class *A Method that uses the information inherited from another class.
		Description: An example of a class is shown on the **top left** shows the Employee class. **Top right** shows the Manager class which inherits from the Employee Class. In the **middle** is an example of an object in the inherited class, a new manager named Joe, which shows the Employee information inherited as well as the department which is a component of the Manager class. At the **bottom** is an example of a method in Manager that uses information inherited from the Employee class, such as getName and setName.

Employee.java

```java
package Staff;

public abstract class Employee {

    private String name;
    private String nINumber;
    private double salary;

    public Employee(String name, String nINumber, double salary) {
        this.name = name;
        this.nINumber = nINumber;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        if (name != null) {
            if (name != "") {
                this.name = name;
            }
        }
    }
```

Employee.java Manager.java

```java
package Staff.management;

        import Staff.Employee;

public class Manager extends Employee {
    private String deptName;

    public Manager(String name, String nINumber, double salary, String deptName) {
        super(name, nINumber, salary);
        this.deptName = deptName;
    }

    public String getDeptName() {
        return deptName;
    }

    public void setDeptName(String deptName) {
        this.deptName = deptName;
    }
}
```

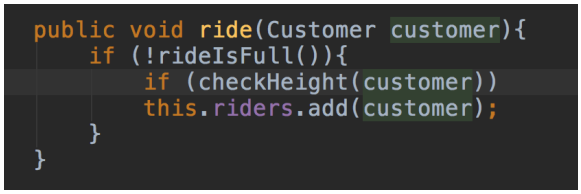
Employee.java Manager.java Developer.java DatabaseAdmin.java

```java
import Staff.management.Manager;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class ManagerTest {
    Manager manager;

    @Before
    public void setUp() throws Exception {
        manager = new Manager("Joe", "AB123456A", 20000.50,"Payroll");
    }

    @Test
    public void canGetName() {
        assertEquals("Joe", manager.getName());
    }

    @Test
    public void canSetName() {
        manager.setName("Sue");
        assertEquals("Sue", manager.getName());
    }
```

```java
    @Before
    public void setUp() throws Exception {
        manager = new Manager("Joe", "AB123456A", 20000.50,"Payroll");
    }
```

```java
    @Test
    public void canGetName() {
        assertEquals("Joe", manager.getName());
    }

    @Test
    public void canSetName() {
        manager.setName("Sue");
        assertEquals("Sue", manager.getName());
    }
```

Unit	Ref	Evidence
P	P.9	Select two algorithms you have written (NOT the group project). Take a screenshot of each and write a short statement on why you have chosen to use those algorithms.
		Description: Top is my theme park's ride algorithm. I chose to use this algorithm as a way of making all the necessary checks that need to be done before adding a customer to a ride. Before a customer can get on it needs to be determined first that there is space on the ride for them, then that they meet the height requirements and then the process of riding can happen. This algorithm is found in the Rides class and is inherited into each type of ride (e.g. roller coaster, ferris wheel, etc). At the **bottom** is an algorithm that searches for a plane. It starts off with no plane, takes in what plane is being searched for, then compares the inputted plane against the planes available, and will return the result as planeFound. I chose to use this algorithm as it helps to carry out a search needed for other methods and algorithms to implement - such as adding planeFound to a new Flight, etc. Both the algorithms included, implement and use methods as well as carry out a search or a check before reaching their final step or break point.



```
public void ride(Customer customer){
    if (!rideIsFull()){
        if (checkHeight(customer))
        this.riders.add(customer);
    }
}
```

```
public Plane getPlane(Plane planeType){
    Plane planeFound = null;
    for (Plane plane : this.planes) {
        if (plane.getPlaneType().equals(planeType)){
            planeFound = this.planes.remove(this.planes.indexOf(plane));
            break;
        }
    }
    return planeFound;
}
```