

# PDA Evidence - Chris Burn, G2 Cohort

## Week 2:

Demonstrate the use of an array in a program. Take screenshots of:

- \*An array in a program
- \*A function that uses the array

Two Arrays or strings, pulled into “add\_length\_of\_arrays” method



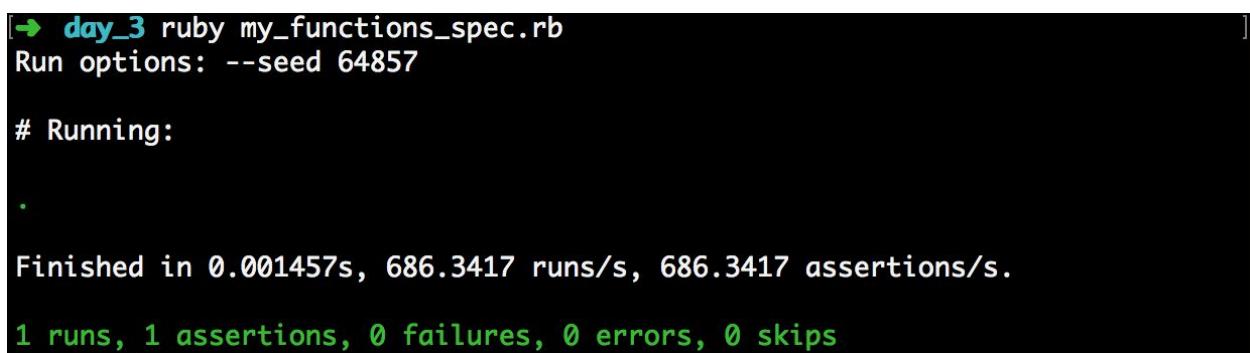
```
my_functions.rb
1 array_1 = [ "Ally", "John" ],
2 array_2 = [ "Steve", "Alan", "Adam" ]
3
4 def add_length_of_arrays(array_1, array_2)
5   add_length_of_arrays = array_1.length() + array_2.length()
6   return add_length_of_arrays
7 end
8
```

\*The result of the function running

Shows TDD testing of method through MiniTest and result of MiniTest in Terminal...



```
my_functions_spec.rb
1 require( 'minitest/autorun' )
2 require( 'minitest/rg' )
3 require_relative( 'my_functions' )
4
5 class My_Functions < MiniTest::Test
6
7
8   def test_add_length_of_arrays
9     result = add_length_of_arrays( [ "Ally", "John" ], [ "Steve", "Alan", "Adam" ] )
10    assert_equal( 5, result )
11  end
12
```



```
[→ day_3 ruby my_functions_spec.rb
Run options: --seed 64857

# Running:

.

Finished in 0.001457s, 686.3417 runs/s, 686.3417 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips]
```

Demonstrate the use of a hash in a program. Take screenshots of:

\*A hash in a program

WITH

\*The result of the function running

Hash and an Array with in-built Hashes, WITH testing of function (separate) via MiniTest



```
pet_shop_spec.rb • pet_shop.rb *
```

```
def setup

  @customers = [ #Array with Hashes
    {
      name: "Craig",
      pets: [],
      cash: 1000
    },
    {
      name: "Zsolt",
      pets: [],
      cash: 50
    }
  ]

  @new_pet = { #Simple Hash
    name: "Bors the Younger",
    pet_type: :cat,
    breed: "Cornish Rex",
    price: 100
  }
end

def test_add_pet_to_customer
  customer = @customers[0]
  add_pet_to_customer(customer, @new_pet) #already defined cust[0]
  assert_equal(1, customer_pet_count(customer))
end
```

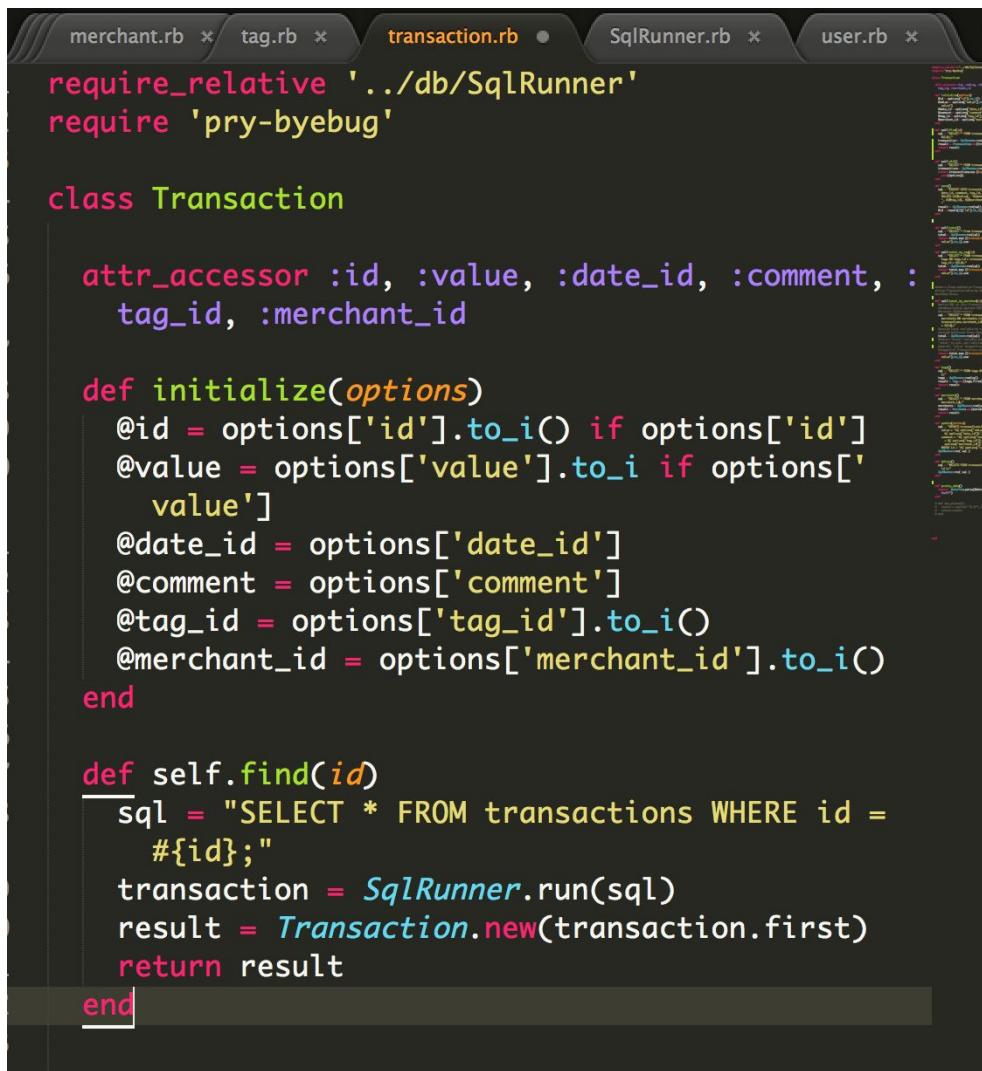
\*A function that uses the hash

```
def add_pet_to_customer(customers, new_pet)
  customers[:pets] << new_pet
end
```

### Week 3:

Demonstrate searching data in a program. Take screenshots of:

\*Function that searches data



A screenshot of a code editor showing the `transaction.rb` file. The file contains Ruby code defining a `Transaction` class with attributes and methods for initializing and finding transactions by ID.

```
require_relative '../db/SqlRunner'
require 'pry-byebug'

class Transaction

  attr_accessor :id, :value, :date_id, :comment, :tag_id, :merchant_id

  def initialize(options)
    @id = options['id'].to_i() if options['id']
    @value = options['value'].to_i if options['value']
    @date_id = options['date_id']
    @comment = options['comment']
    @tag_id = options['tag_id'].to_i()
    @merchant_id = options['merchant_id'].to_i()
  end

  def self.find(id)
    sql = "SELECT * FROM transactions WHERE id = #{id};"
    transaction = SqlRunner.run(sql)
    result = Transaction.new(transaction.first)
    return result
  end
end
```

\*The result of the function running

```
moneyapp=# SELECT * FROM transactions WHERE id = 1;
+----+----+----+----+----+
| id | value | date_id | comment | tag_id | merchant_id |
+----+----+----+----+----+
| 1  | 30   | 2017-06-04 | Weekly shop. | 1 | 2
+----+----+----+----+----+
(1 row)
```

Demonstrate sorting data in a program. Take screenshots of:

\*Function that sorts data

```
require_relative '../db/SqlRunner'
require 'pry-byebug'

class Transaction

  attr_accessor :id, :value, :date_id, :comment, :
    tag_id, :merchant_id

  def initialize(options)
    @id = options['id'].to_i() if options['id']
    @value = options['value'].to_i if options['
      value']
    @date_id = options['date_id']
    @comment = options['comment']
    @tag_id = options['tag_id'].to_i()
    @merchant_id = options['merchant_id'].to_i()
  end

  def self.sort_by_date
    sql = "SELECT * FROM transactions ORDER BY
      date_id;_
  end

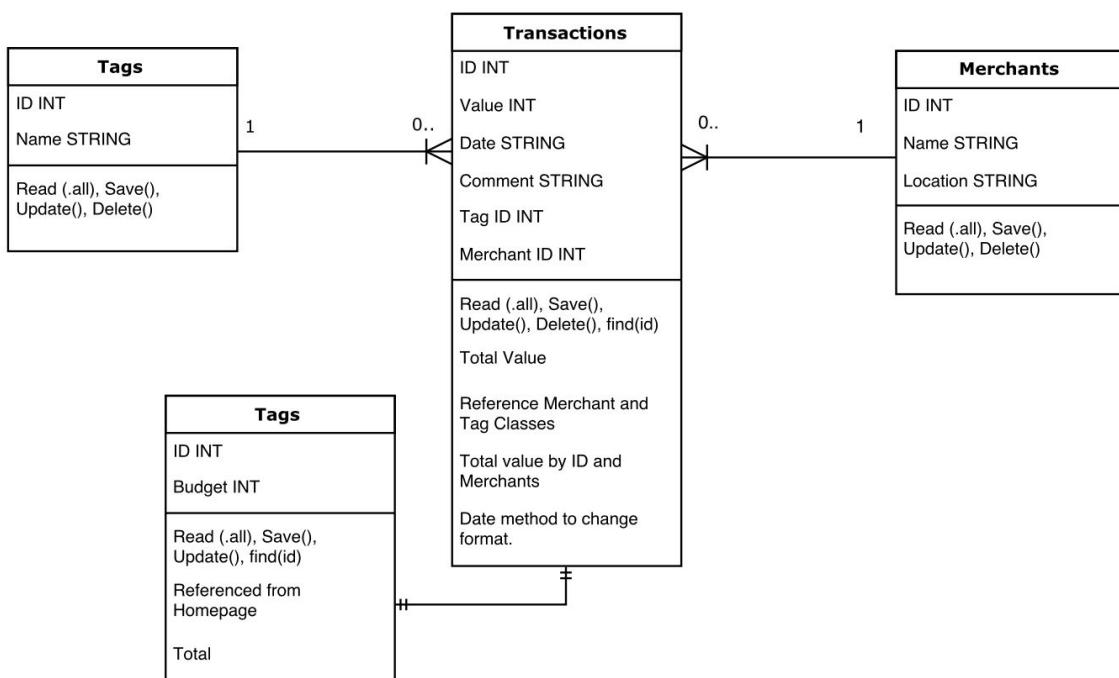
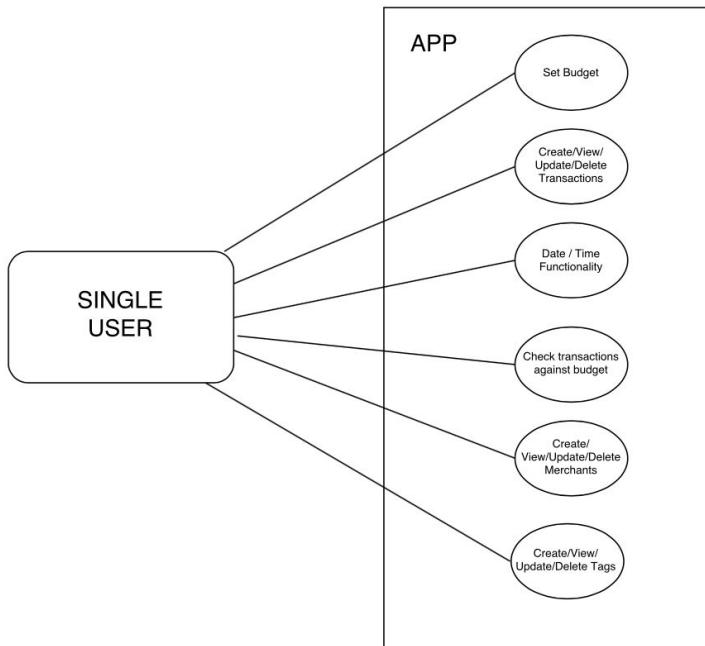
```

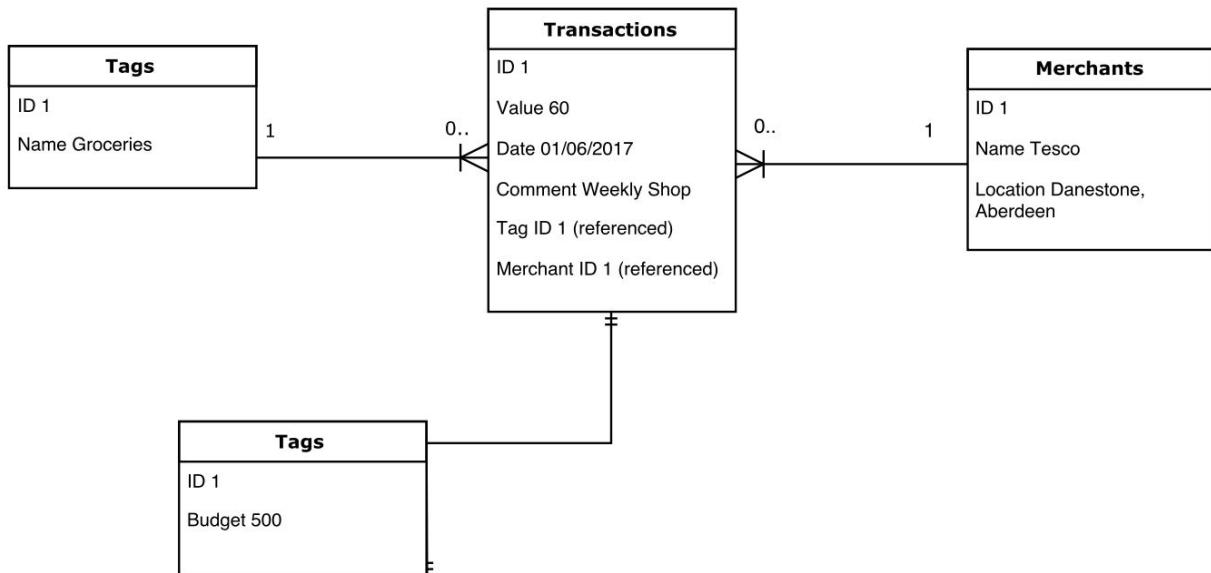
\*The result of the function running

	id	value	date_id	comment	tag_id	merchant_id
	5	39	2017-05-20	Train tickets to Glasgow. 7	6	
	4	50	2017-05-21	Night out with mates. 5	7	
	3	15	2017-05-27	Mid-week takeaway. 4	4	
	7	40	2017-05-28	CDs and Blu-Ray for sisters birthday. 6	2	
	10	50	2017-05-29	Whisky for me! 9	9	
	6	45	2017-05-30	Weekly shop. 1	1	
	8	70	2017-06-01	Paint for home decoration. 3	3	
	1	30	2017-06-04	Weekly shop. 2	1	
	2	19	2017-06-05	Replacement window wipers for car. 8	5	
	11	25	2017-06-09	More pizza! 4	4	
	9	200	2017-06-11	Flights to Madrid 10	6	

(11 rows)

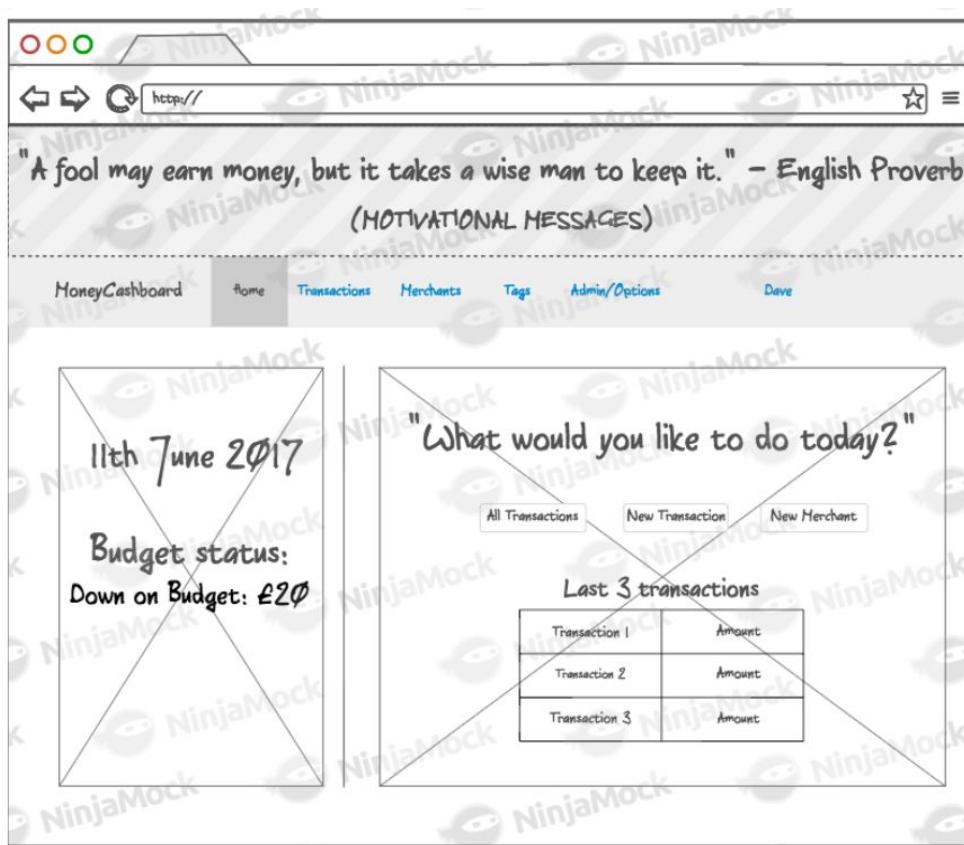
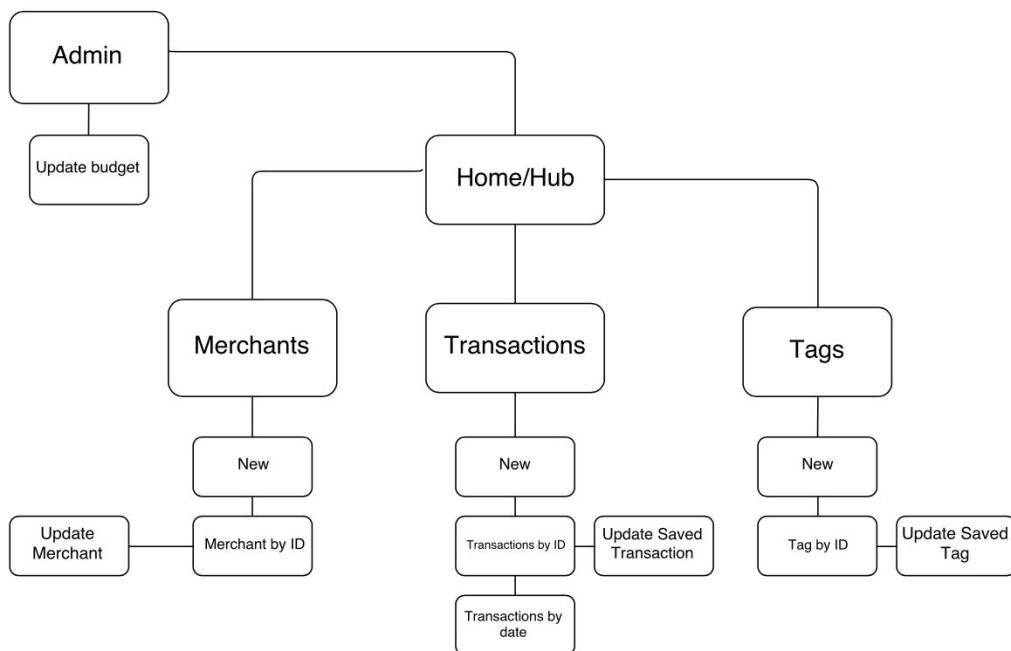
## Week 5:





Money CashBoard Implementation Constraints Plan - Sheet1

IMPLEMENTATION CONSTRAINTS PLAN		
Constraint	Possible Effect on Constraint on Product	Solution
Hardware and Software Problems	Reduced user-experience cross platforms, and on different user machines.	Program to compatible cross browsers and machines including mobile platforms. Use of dynamic displays using FlexBox should help to achieve this.
Performance Requirements	Reduced performance when frequently calling on database data. Potential for reduced user experience.	Unlikely to have mass traffic as it's an application for personal use. Data being called from database is of relatively small size and is unlikely to cause an issue with performance.
Persistent Storage and Transactions	Potential for future loss or reduction of free/cheap database capacity may curtail use of application going forward.	To get around this the website design will avoid the use of large media files / graphics /styling which also has potential for an upside on meeting budget requirements.
Usability	User unable to navigate site with ease, reducing user experience and reduce likelihood of future use and roll out.	Personal use of application, not being used in public. Simple design with large menus/buttons and bold text and colours should reduce potential for navigation errors.
Budgets	No budget considered but in theory a limited budget will curtail use of premium servers meaning potential for slower performance and risk of downtime. In addition, use of non premium graphics and design may limit the visual appeal of the finished product.	Use of free web tooling and server space will keep web app cost minimal for as long a period of time as application is required.
Time Limitations	6 day turnaround for finished product will mean limited time to perfect usability and design/finish of product. Additional functionality above MVP may be limited.	Careful planning of project should reduce potential for road-blocks in execution and possible delays to project.



The saving man becomes the free man. — Proverb  
(MOTIVATIONAL MESSAGES)

MoneyCastboard    Home    Transactions    Merchants    Tags    Admin/Options    Dave

**ALL TRANSACTIONS**

ID	Merchant	Location	Amount	Date	Tag
1	Asda	Garthdee, Aberdeen	£34.50	01/06/17	Groceries
2	Tesco	Danestone, Aberdeen	£50.00	26/05/17	Groceries
3	B&Q	Online	£25.00	24/05/17	DIY
4	Domino's	Online	£20.00	22/05/17	Take-away
5	Brewdog	Union St, Aberdeen	£30.00	22/05/17	Social

New Transaction    Home

Pseudo-code example showing a method calling on the result of an INNER JOIN on two database tables:

```
#make a Class method on Transaction Class to
display Transaction value by Merchant from
Merchant Class.

def self.total_by_merchant(id)
  #write SQL to join Transaction and Merchant
  #database tables against ID(Transaction) and
  #Merchant ID(Merchant)
  sql = "SELECT * FROM transactions INNER JOIN
    merchants ON merchants.id =
    transactions.merchant_id WHERE merchant_id
    = #{id};"
  #assign local variable to running of SQL code
  #through SqlRunner Class Ruby file
  total = SqlRunner.run(sql)
  #return "total" variable and 'map'/loop
  #'total' to pull out individual transactions
  #sum all 'value' properties (converted to
  #Integer) of Transactions called
  return total.map{|transaction| transaction['
    value'].to_i}.sum
end
```

**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

**Show an interaction with data persistence. Take a screenshot of:**

- \* Data being inputted into your program
- \* Confirmation of the data being saved

**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

The screenshot shows a web browser window at localhost:4567/merchant/new. The page title is "Welcome to RealityCheck!". Below the title is a large image of a pink piggy bank. A quote is displayed: "*A fool may earn money, but it takes a wise man to keep it*" - English Proverb. At the bottom of the page is a navigation bar with links for Home, Transactions, Merchants, Tags, and Budget. Below the navigation bar is a form titled "Add New Merchant:" with fields for Name (Brewdog) and Location (Elton), and a Save button.

```
get '/merchant/new' do
  erb(:merchant_new)
end

post '/merchants' do
  @merchant = Merchant.new(params)
  @merchant.save()
  erb(:merchant_create)
end
```

```
controller.rb ✘ merchant_new.erb ✘ console.rb ✘ styles.css ✘ b ✘
1 <div class="cell">
2
3 <h3>Add New Merchant:</h3>
4
5 <form action="/merchants" method="POST">
6
7 <div>
8 <label for="name">Name:</label>
9   <input type="text" id="name" name="name" required>
10 </div>
11 <div>
12 <label for="location">Location:</label>
13   <input type="text" id="location" name="location" required>
14 </div>
15 <input type="submit" value="Save">
16
17 </form>
18 </div>
19
20
21
```

```
controller.rb ✘ merchant_create.erb ✘ console.rb ✘ styles.css ✘ x
1 <div class="cell">
2 <h3>Merchant saved!</h3>
3
4 <h3><a href="/merchants">Back to Merchants</a></h3>
5
6 <h3><a href="/transaction/new">Back to New Transaction</a></h3>
7 </div>
8
```

```
sty merchant.rb × tag.rb × transaction.rb × SqlRunner.rb × rb ×
require_relative '../db/SqlRunner'
require 'pry-byebug'

class Merchant

attr_accessor :id, :name, :location

def initialize(options)
  @id = options['id'].to_i if options['id']
  @name = options['name']
  @location = options['location']
end

def self.all()
  sql = "SELECT * FROM merchants;"
  merchants = SqlRunner.run(sql)
  return merchants.map{|options| Merchant.new(
    options)}
end

def save()
  sql = "INSERT INTO merchants (name, location)
        VALUES ('#{@name}', '#{@location}')
        RETURNING *;"
  result = SqlRunner.run(sql)
  @id = result[0]['id'].to_i()
end
```

```
sty merchant.rb × tag.rb × transaction.rb × SqlRunner.rb × rb ×
result = SqlRunner.run(sql)
@id = result[0]['id'].to_i()
end

def self.find(id)
  sql = "SELECT * FROM merchants WHERE id = #{id}"
  ;
  merchant = SqlRunner.run(sql)
  result = Merchant.new(merchant.first)
  return result
end

def update(options)
  sql = "UPDATE merchants SET
        name = '#{ options['name'] }',
        location = '#{ options['location'] }'
        WHERE id = '#{ options['id'] }';"
  SqlRunner.run( sql )
end

def delete()
  sql = "DELETE FROM merchants WHERE id=#{ @id }";
  "
  SqlRunner.run( sql )
end

end
```

```

17 | Amazon UK      | Online
18 | Scotrail       | Train Station, Aberdeen
19 | Halfords        | Berryden, Aberdeen
20 | Whisky Exchange | Online
21 | British Airways | Online
22 | Falkirk Council | West Bridge Street, Falkirk
23 | Asda            | Garthdee, Aberdeen
24 | Tesco           | Danestone, Aberdeen
25 | B&Q             | Online
26 | Dominos          | Online
27 | Brewdog          | Merchant City, Glasgow
28 | Amazon UK        | Online
29 | Scotrail         | Train Station, Aberdeen
30 | Halfords          | Berryden, Aberdeen
31 | Whisky Exchange   | Online
32 | British Airways    | Online
33 | Falkirk Council   | West Bridge Street, Falkirk
34 | Brewdog           | Ellon
(34 rows)

```

(END)

localhost:4567/merchants



Welcome to RealityCheck!

*"A fool may earn money, but it takes a wise man to keep it" - English Proverb*

Merchant saved!

[Back to Merchants](#)

[Back to New Transaction](#)

localhost:4567/merchants

Current spend by Merchant: £50

[Merchant Profile](#)

---

British Airways

Location: Online

Current spend by Merchant: £200

[Merchant Profile](#)

---

Falkirk Council

Location: West Bridge Street, Falkirk

Current spend by Merchant: £0

[Merchant Profile](#)

---

Brewdog

Location: Ellon

Current spend by Merchant: £0

[Merchant Profile](#)

## Week 6:

Demonstrate the use of Polymorphism in a program:

### POLYMORPHISM

- Capacity to take on different forms.
  - A language's ability to process objects of various types and classes through a single, uniform interface.
  - Example, how a parent class refers to a child class object
  - Any object that satisfies more than one IS-A relationship is polymorphic in nature.
  - Fair to say that every Object in Java is polymorphic in nature, each one passes IS-A test for itself and for Object class.
- E.g. Animal class, Cat IS sub-class of Animal. Cat satisfies IS-A relationship of own type and its Super-class, Animal.

```
package com.example.user.multipleclasshomework;
import java.util.IllegalFormatException;

/**
 * Created by user on 20/06/2017.
 */
public class Employee {
    private int empId;
    private String name;
    private String ssn;
    private double salary;

    public Employee(int empId, String name, String ssn, double salary) {
        this.empId = empId;
        this.name = name;
        this.ssn = ssn;
        this.salary = salary;
    }

    public int getEmpId() {
        return empId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        if(name != null && name != "") {
```

```
package com.example.user.multipleclasshomework;

/**
 * Created by user on 20/06/2017.
 */
public class Manager extends Employee {

    private String deptName;

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

    public String getDeptName() {
        return this.deptName;
    }
}
```

```

package com.example.user.multipleclasshomework;

/*
 * Created by user on 20/06/2017.
 */

public class Director extends Manager {
    private double budget;

    public Director(int empId, String name, String ssn, double salary, String deptName, double budget) {
        super(empId, name, ssn, salary, deptName);
        this.budget = budget;
    }

    public double getBudget(){
        return this.budget;
    }
}

import org.junit.Test;
import static org.junit.Assert.assertEquals;

/*
 * Created by user on 20/06/2017.
 */

public class TestDirector {

    Director director;

    @Before
    public void before(){
        director = new Director(20, "Sean Jones", "E9H1J1", 50000.00, "Sausages", 500000.00);
    }

    @Test
    public void canRaiseSalary(){
        director.raiseSalary(80000.00);
        assertEquals(80000.00, director.getSalary(), 0.01);
    }

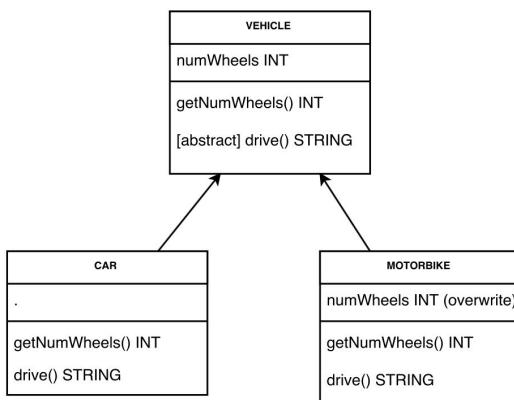
    @Test
    public void canSetName(){
        director.setName("Brent Jones");
        assertEquals("Brent Jones", director.getName());
    }
}

```

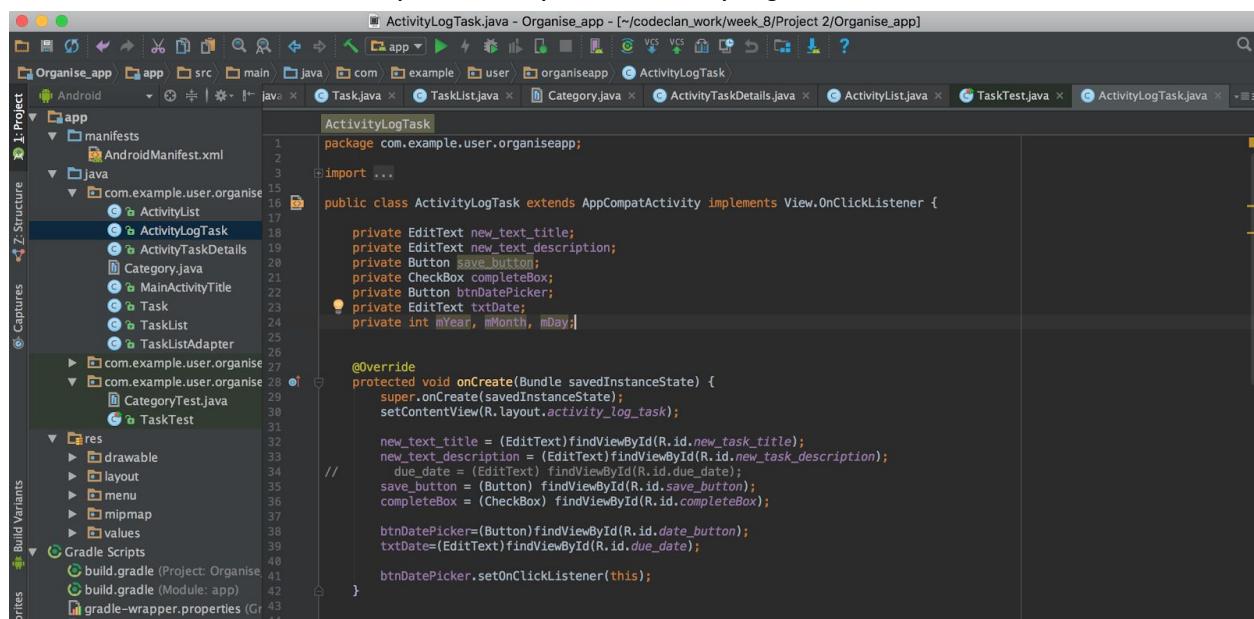
All 2 tests passed - 5ms

## Week 7:

### An Inheritance Diagram



*Take a screenshot of an example of encapsulation in a program*



The screenshot shows the Android Studio interface with the project 'Organise\_app' open. The 'ActivityLogTask.java' file is selected in the editor. The code implements encapsulation by using private fields and public getters and setters. The code is as follows:

```
package com.example.user.organiseapp;
import ...;
public class ActivityLogTask extends AppCompatActivity implements View.OnClickListener {
    private EditText new_text_title;
    private EditText new_text_description;
    private Button save_button;
    private CheckBox completeBox;
    private Button btnDatePicker;
    private EditText txtDate;
    private int mYear, mMonth, mDay;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_log_task);
        new_text_title = (EditText) findViewById(R.id.new_task_title);
        new_text_description = (EditText) findViewById(R.id.new_task_description);
        due_date = (EditText) findViewById(R.id.due_date);
        save_button = (Button) findViewById(R.id.save_button);
        completeBox = (CheckBox) findViewById(R.id.completeBox);
        btnDatePicker=(Button)findViewById(R.id.date_button);
        txtDate=(EditText)findViewById(R.id.due_date);
        btnDatePicker.setOnClickListener(this);
    }
}
```

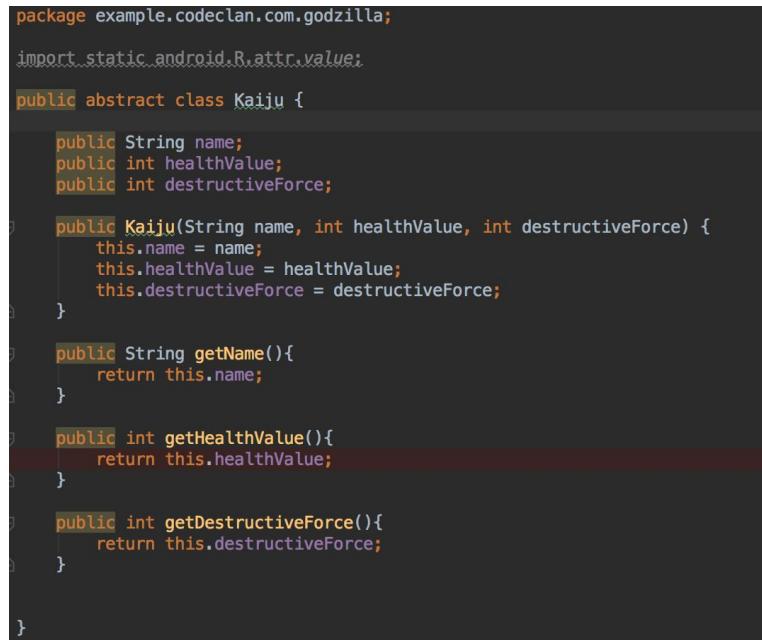
*Take a screenshot of the use of Inheritance in a program. Take screenshots of:*

*\*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.*



The screenshot shows a Java code snippet illustrating inheritance and encapsulation. The code defines an abstract class 'Kaiju' with private fields and public getters and setters. The code is as follows:

```
package example.codeclan.com.godzilla;
import static android.R.attr.value;
public abstract class Kaiju {
    public String name;
    public int healthValue;
    public int destructiveForce;
    public Kaiju(String name, int healthValue, int destructiveForce) {
        this.name = name;
        this.healthValue = healthValue;
        this.destructiveForce = destructiveForce;
    }
    public String getName(){
        return this.name;
    }
    public int getHealthValue(){
        return this.healthValue;
    }
    public int getDestructiveForce(){
        return this.destructiveForce;
    }
}
```

```
package example.codeclan.com.godzilla;

/**
 * Created by user on 21/06/2017.
 */

public class Godzilla extends Kaiju implements Attack {

    public Godzilla(String name, int healthValue, int destructiveForce) {
        super(name, healthValue, destructiveForce);
    }

    /**
     * Roar method
     */
    public String roar(){
        return "ROAR!";
    }
}
```

```
package example.codeclan.com.godzilla;

/**
 * Created by user on 21/06/2017.
 */

public class Pikachu extends Kaiju implements Attack {

    public Pikachu(String name, int healthValue, int destructiveForce){
        super(name, healthValue, destructiveForce);
    }

    public String attack(){
        return "Kablammo!!";
    }

    public String roar(){
        return "PIKA! PIKA!";
    }
}
```

```
package example.codeclan.com.godzilla;

/**
 * Created by user on 21/06/2017.
 */

public class Runner {

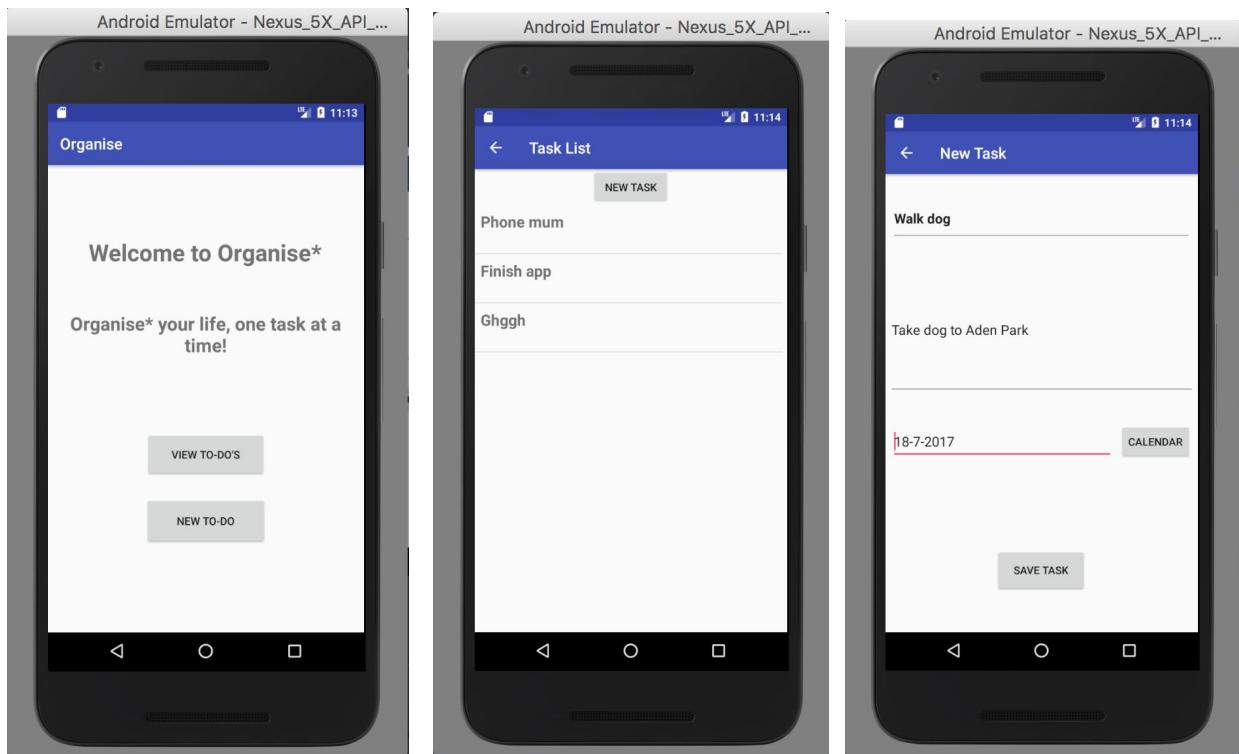
    public static void main(String[] args){

        Godzilla godzilla = new Godzilla("Zilla", 120, 50);
        Pikachu pikachu = new Pikachu("Pika", 90, 35);
        SkyScraperskyScraperscraper = new SkyScraperscraper("largebuilding", 30);
        Townhall townhall = new Townhall("smallbuilding", 20);

        System.out.println("Zilla enters the City, looks like trouble!");
        System.out.println("Zilla looks pretty healthy with health " + godzilla.getHealthValue());
    }
}
```

*Take a screenshot of one of your projects where you have worked alone and attach the Github link.*

[https://github.com/chris-burn/To-Do\\_list\\_project](https://github.com/chris-burn/To-Do_list_project)



Take screenshots or photos of your planning and the different stages of development to show changes.

## Android App, PROJECT PLANNING:

**Organise\* - Android Project**

**SETUP**

- Site Map
- Case Diagram
- Class Diagram
- Object Diagram
- Activity Diagram
- Implementation Constraints Plan
- Wireframe Diagrams
- Database Table Diagram - N/A
- Inheritance Diagram - N/A

**Must - MVP**

- Functional app interface in Android environment
- READ all saved tasks in list view
- LOG/SAVE new tasks
- READ full task details in separate screen
- Mark tasks as complete

**Should - Basic Extensions to add functionality**

- SET due dates on tasks
- Addition of Home / Welcome page
- Visual elements of App - Theme's etc
- Pop-up information upon Action (task save, update, delete etc) - Toast or Snackbar
- Addition of CATALOGY when logging tasks

**Organise\* - Android Project**

**SETUP**

- Site Map
- Case Diagram
- Class Diagram
- Object Diagram
- Activity Diagram
- Implementation Constraints Plan
- Wireframe Diagrams
- Database Table Diagram - N/A
- Inheritance Diagram - N/A

**Must - MVP**

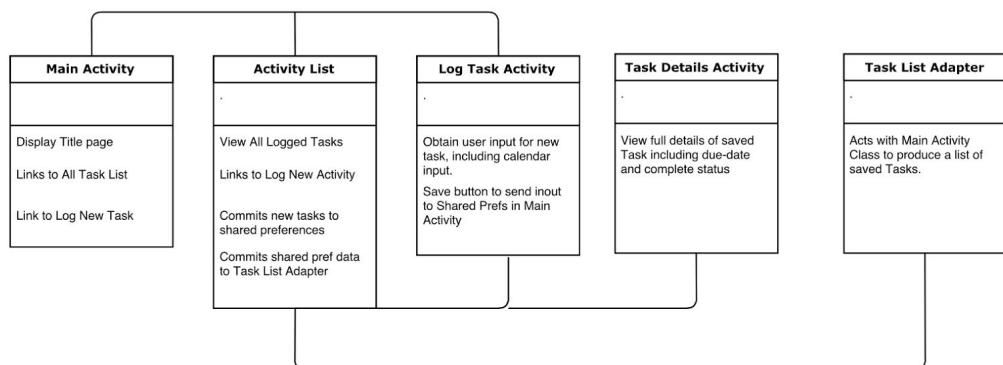
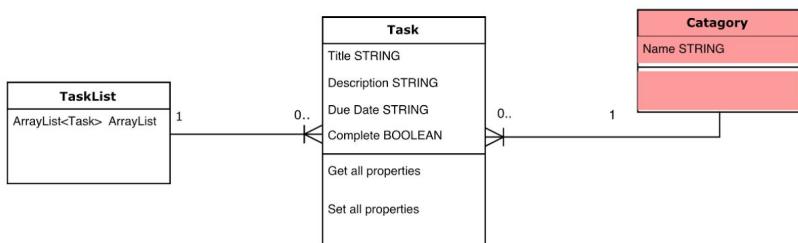
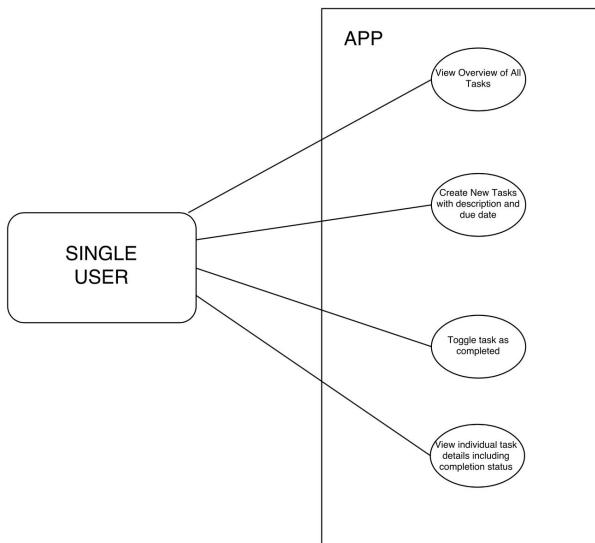
- Functional app interface in Android environment
- READ all saved tasks in list view
- LOG/SAVE new tasks
- READ full task details in separate screen
- Mark tasks as complete

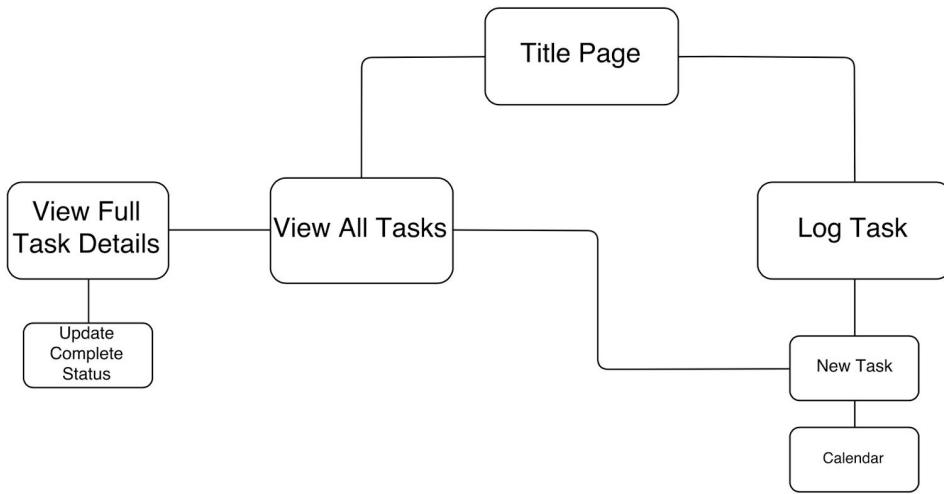
**Should - Basic Extensions to add functionality**

- SET due dates on tasks
- Addition of Home / Welcome page
- Visual elements of App - Theme's etc
- Pop-up information upon Action (task save, update, delete etc) - Toast or Snackbar
- Addition of CATALOGY when logging tasks

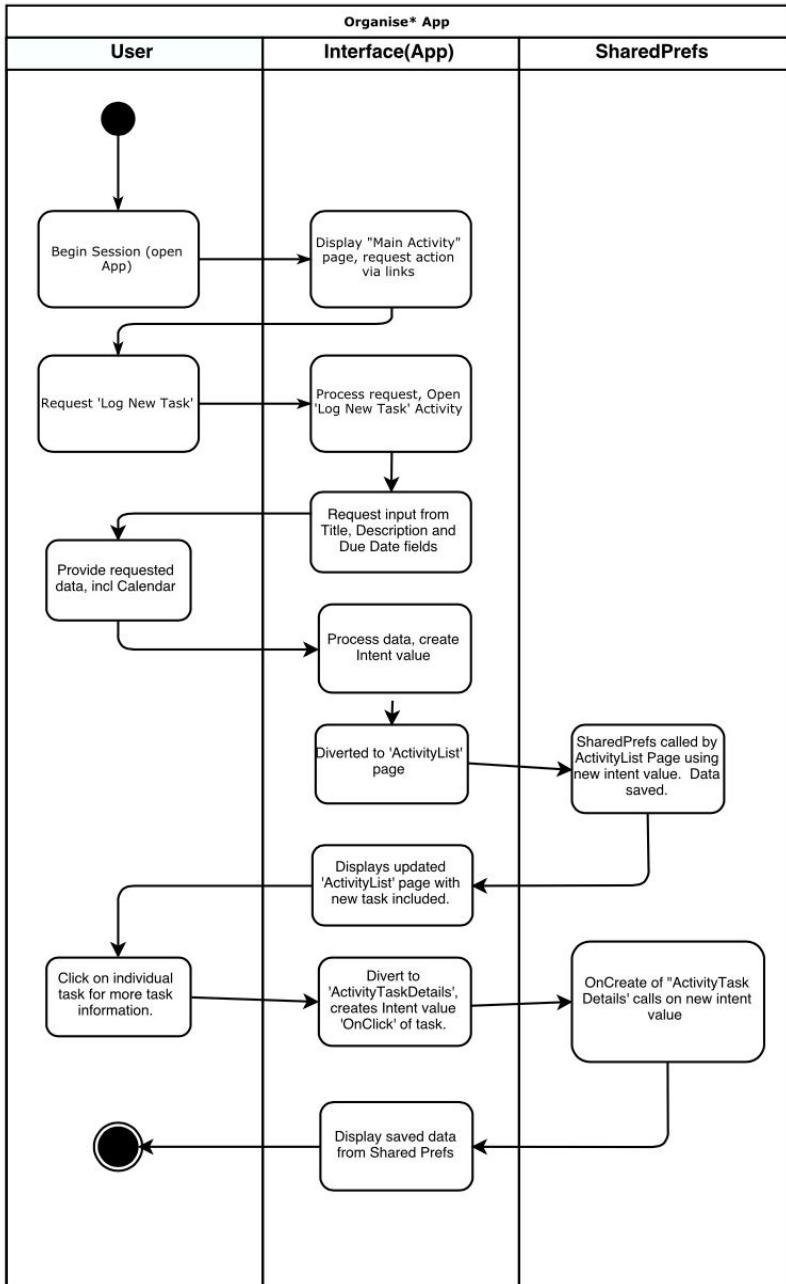
**Would - future development**

- DELETE tasks
- UPDATE tasks
- View tasks sorted by Due Date
- View Completed tasks (filter by complete)
- Separate activity screen, View by Category
- Separate viewing of Priority (Starred) tasks





IMPLEMENTATION CONSTRAINTS PLAN		
Constraint	Possible Effect on Constraint on Product	Solution
Hardware and Software Problems	Limited use - Android App only	Multiple versions of apps would be required. No instant solution.
Performance Requirements	Potential limit to performance especially as the data size increases. Minimal visuals in app at present should limit issues with regards app performance.	Unlikely to have mass traffic as it's an application for personal use. Data being used for App use is of relatively small size and is unlikely to cause an issue with performance. Use of SQL database would solve any data volume concerns.
Persistent Storage and Transactions	Data persistence achieved through SharedPreferences internal to app only, no database use which may limit app potential.	To get around this the website design will avoid the use of large media files / graphics /styling which also has potential for an upside on meeting budget requirements.
Usability	User unable to navigate site with ease, reducing user experience and reduce likelihood of future use and roll out.	Simple design layout with large menus/buttons and bold text and colours should reduce potential for navigation errors. Personal use of application, not for public distribution in current state.
Budgets	No budget considered but in theory a limited budget will curtail use of premium servers meaning potential for slower performance and risk of downtime. In addition, use of non premium graphics and design may limit the visual appeal of the finished product.	Use of free web tooling and server space will keep web app cost minimal for as long a period of time as application is required.
Time Limitations	6 day turnaround for finished product means limited design/finish of product. Additional functionality above MVP may be limited.	Careful planning of project should reduce potential for road-blocks in execution and possible delays to project.





## Week 13: Algorithms

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.

Weekend homework (Blackjack):

This method populates a 'deck' (ArrayList) with card types and numbers via a nested for loop which loops through a designated set of ENUMS for card type (suits) and loads the ArrayList with a connected set of ENUMS for card numbers. This means for each card type there are a set of card numbers. I.e. Hearts has an ArrayList of (Ace, 2, 3, 4 ... King), as does Diamonds and Clubs. The result is an ArrayList of 52 cards (4 \*13) which makes up the playing deck for the game of Blackjack.

```
public void populateDeck() {
    for (CardSuit type : CardSuit.values()){
        for (CardNumber number : CardNumber.values()){
            deck.add(new Card(number, type));
        }
    }
    Collections.shuffle(deck);
}
```

Weekend homework (WordCount Android App, homework):

This method is used to calculate the length of a String input from an EditText box. Setting a counter at the start of the method via the ‘words’ variable, the for loop iterates over the String and splits the string into separate characters via the split function. It then increments the word count for each input of a blank entry. The words variable is then converted to a String via another variable designation as required by the String method output.

```
public String getWordCount(String input){  
    int words = 0;  
    for (String string : input.split(" ")){  
        words += 1;  
    }  
    int number = words;  
    String numberToString = Integer.toString(number);  
    return numberToString;  
}
```