

# 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.

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

Each point requires details that cover each element of the Assessment Criteria, along with a brief description of the kind of things you should be showing.

## Week 1

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	

```
@customers = [  
  {  
    name: "Alice",  
    pets: [],  
    cash: 1000  
  },  
  {  
    name: "Bob",  
    pets: [],  
    cash: 50  
  },  
  {  
    name: "Jack",  
    pets: [],  
    cash: 100  
  }  
]
```

Paste Screenshot here

Description here

This screenshot shows the @customers hash being created in order to initialize the test spec.

```
73 # 11. check customer cash
74 def customer_cash(customer)
75   return customer[:cash]
76 end
```

#### Paste Screenshot here

#### Description here

The hash **@customer** being created in the previous screenshot is being passed into the function here with an index. This function is taking the value of **cash** for the customer specified in the index and returning it.

#### Paste

```
def test_customer_cash
  cash = customer_cash(@customers[0])
  assert_equal(1000, cash)
end
```

#### Screenshot here

```
→ start_point git:(wlwehw) x ruby specs/pet_shop_spec.rb
Run options: --seed 52761

# Running:

*

Finished in 0.002516s, 397.4563 runs/s, 397.4563 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
→ start_point git:(wlwehw) x
```

#### Description here

The above images show the function in which the hash is used, and the result when run. The function tests that the value of **cash** for the first customer (index 0) is correct (1000).

## 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	

```
@song1 = Song.new("Mr. Brightside", "The Killers")
@song2 = Song.new("Purple Rain", "Prince")
@song3 = Song.new("Like a Prayer", "Madonna")
@song4 = Song.new("Let's Get It On", "Marvin Gaye")
@song5 = Song.new("Private Eyes", "Hall & Oates")
@song6 = Song.new("I Want It That Way", "The
Backstreet Boys")
@song7 = Song.new("Born to Run", "Bruce Springsteen")
@song8 = Song.new("I Wanna Dance With Somebody",
"Whitney Houston")

@song_list1 = [@song2, @song4, @song6, @song7]
```

Paste Screenshot here

```
@song1 = Song.new("Mr. Brightside", "The Killers")
@song2 = Song.new("Purple Rain", "Prince")
@song3 = Song.new("Like a Prayer", "Madonna")
@song4 = Song.new("Let's Get It On", "Marvin Gaye")
@song5 = Song.new("Private Eyes", "Hall & Oates")
@song6 = Song.new("I Want It That Way", "The
Backstreet Boys")
@song7 = Song.new("Born to Run", "Bruce Springsteen")
@song8 = Song.new("I Wanna Dance With Somebody",
"Whitney Houston")

@song_list1 = [@song2, @song4, @song6, @song7]
@song_list2 = [@song1, @song3, @song6, @song8]
```

### Description here

This screenshot shows 2 arrays being created out of a selection of 8 song objects created above. I will use **@song\_list2** in the example below.

```
@new_room1 = Room.new(@guest_list1, @song_list2, 10, 5)
```

### Paste Screenshot here

```
def play_song(song_pos)
  unless @song_list[song_pos].nil?
    cheer_text = ''
    @guest_list.each do |guest|
      cheer_text +=
        guest.cheer(@song_list[song_pos].name) unless
        guest.cheer(@song_list[song_pos].name).nil?
    end
    return "#{cheer_text}#{@song_list[song_pos].name} by
    #{@song_list[song_pos].artist} is playing!"
  end
  return nil
end
```

### Description here

This screenshot shows that the barrage @song\_list2 is pulled in as a property of @new\_room1. The function takes a song position as an integer and checks the array (@song\_list2) for the song at the position. It then displays the text, “<song name> by <song artist> is playing!”. It then also loops through each guest in the room and compares the song’s title to the guest’s favourite song. If they match, an additional cheer\_text string (“Wooo!!”) is added to the beginning of the return.

### Paste Screenshot here

```
def test_play_song__pass
  assert_equal(true,
    @new_room1.play_song(0).include?("Mr. Brightside by
    The Killers is playing!"))
end
```

#### Description here

This screenshot shows the test comparing the output of the function to the expected string, and also shows the test passing.

### Week 3

```
→ w2_we_homework git:(master) x ruby specs/room_spec.rb
Run options: --seed 31612

# Running:

.

Finished in 0.001078s, 927.6438 runs/s, 927.6438 assertions/s.

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

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	

#### Paste Screenshot here

```

def customers
  sql = "SELECT c.*
        FROM customers c
        INNER JOIN tickets t
          ON c.id = t.customer_id
        INNER JOIN screenings sfi
          ON s.id = t.screening_id
        WHERE s.film_id = $1"
  values = [@id]
  result = SqlRunner.run(sql, values)
  return result.map{|customer| Customer.new(customer) }
end

```

#### Description here

This is a method on the Film object that returns an array of Customer objects associated with the Film. It does this by querying the database and joining the Tickets and Screenings tables to the Customer table in order to bring back the relevant ID's, and then maps these into new objects.

#### Paste Screenshot here

```

[2] pry(main)> film1.customers
=> [#<Customer:0x007fba19135738 @funds=93, @id=6, @name="Mal">,
    #<Customer:0x007fba191353f0 @funds=53, @id=8, @name="River">,
    #<Customer:0x007fba19134e78 @funds=63, @id=9, @name="Kaylee">]

```

#### Description here

This shows the results of the method 'customers' above. It is an array of Customer objects created from the results of the SQL that ran against the database.

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	

```

def self.all
  sql = "SELECT * FROM films
        ORDER BY title"
  result = SqlRunner.run(sql)
  return result.map { |m| Film.new(m) }
end
end

```

Paste Screenshot here

Description here

This function brings back all the fields from the 'films' table in the database and sorts them by the 'title' field. It then maps the values into new 'film' objects.

```

[1] pry(main)> Film.all
=> [#<Film:0x007f921699c3e8 @id=20, @price=6.5, @title="American Pie">,
    #<Film:0x007f921699c2f8 @id=19, @price=6.0, @title="Fight Club">,
    #<Film:0x007f921699c230 @id=17, @price=7.5, @title="The Matrix">,
    #<Film:0x007f921699c168 @id=18, @price=8.0, @title="The Sixth Sense">]

```

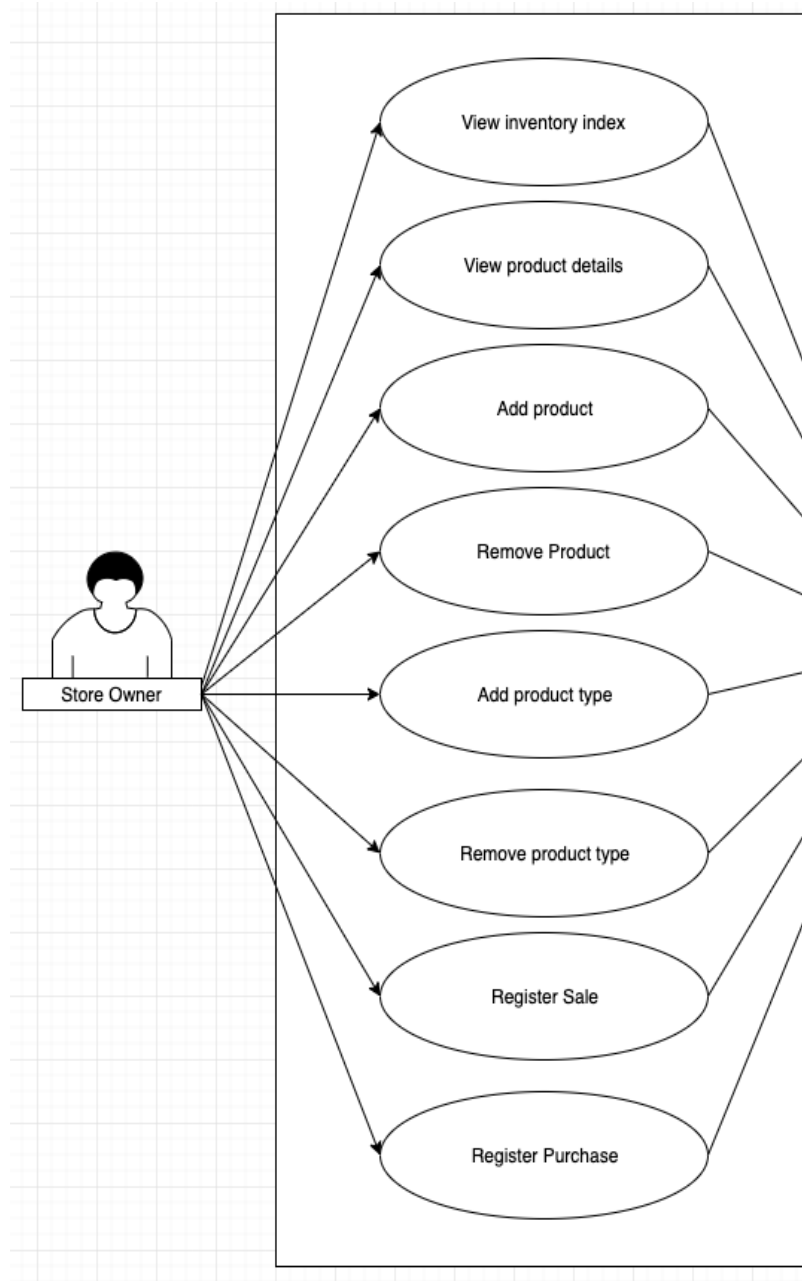
Paste Screenshot here

Description here

The results of the function shows that the objects have been returned into an array in alphabetical order by Title.

Unit	Ref	Evidence	
A&D	A.D.1	A Use Case Diagram	

**Paste Screenshot here**

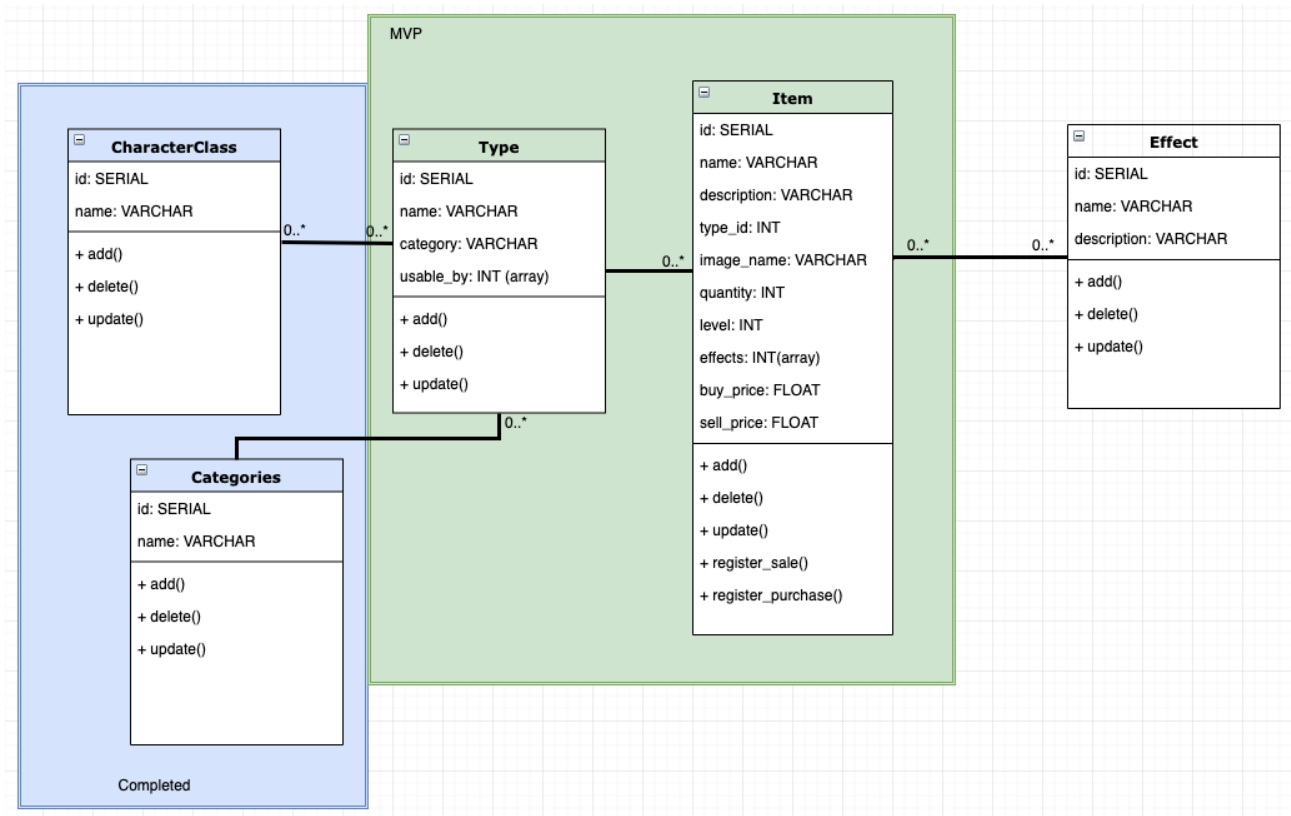


**Description here**

This shows the use case for a shop inventory system that I developed for my project. As a basic inventory management system, the only user would be the store owner, who would need to be able to view the items in stock, drill down to the product details, add and remove products, add and remove product types, and register sales and purchases.



Unit	Ref	Evidence	
A&D	A.D.2	A Class Diagram	

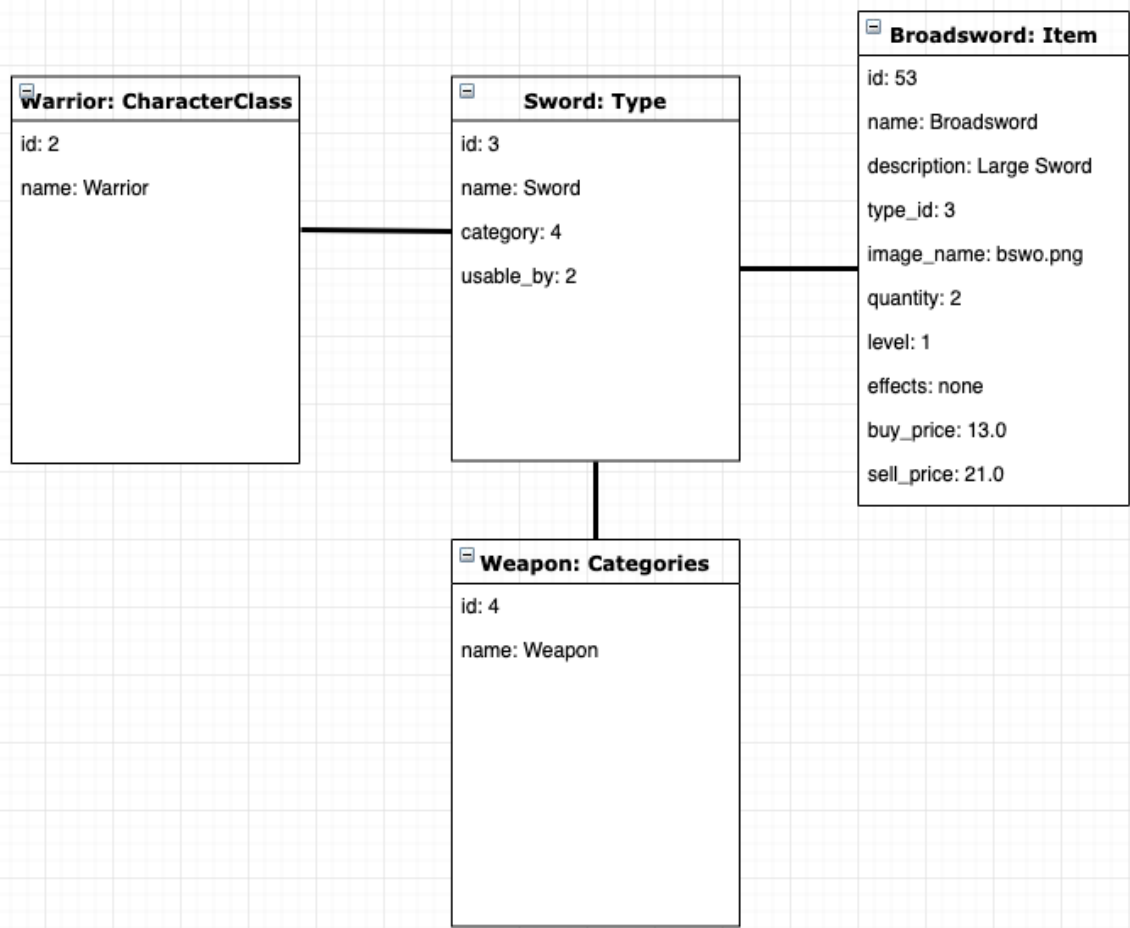


**Paste Screenshot here**

**Description here**

This is the class diagram for my project. The two key classes for the mvp were the Item and Type tables. As an extension, I was also able to add CharacterClass and Categories. As a further extension I had planned to add an Effect class, but did not have time in the end. CharacterClass is a many to many relation to Type, which also required the relationships to be built separately.

Unit	Ref	Evidence	
A&D	A.D.3	An Object Diagram	

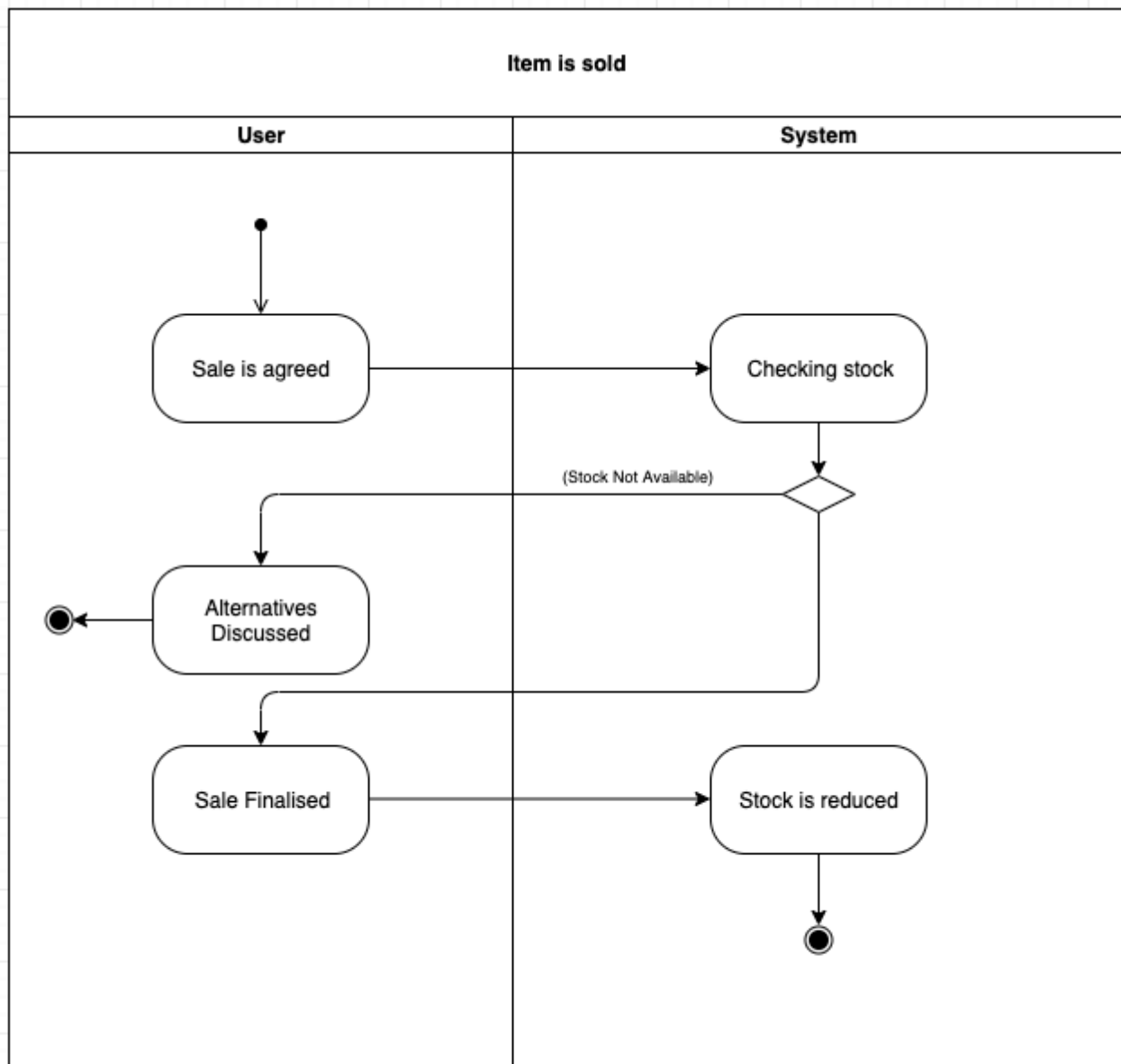


Paste Screenshot here

Description here

This object diagram shows on the right hand side a Broadsword item, which is linked to the Sword type. Sword types are assigned to the category of Weapon. Sword types are usable by the Warrior Character Class.

Unit	Ref	Evidence	
A&D	A.D.4	An Activity Diagram	



**Paste Screenshot here**

**Description here**

The activity diagram above shows the process for selling at item using the stock inventory system. If a sale is agreed, stock must be checked using the app. If stock is available, the sale can be finalised, and the stock level reduced in the app.

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	

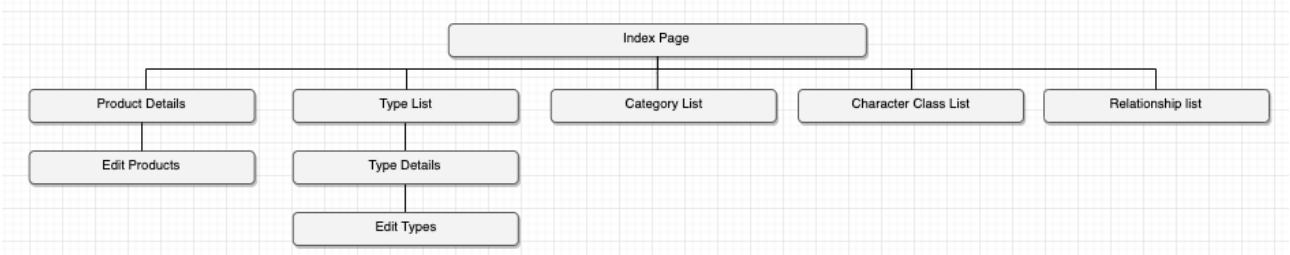
**Paste Screenshot here**

Constraint Category	Implementation Constraint	Solution
Time Limitations	There may not be enough time to complete all the features. This could mean there is no working application by the deadline.	Restrict the features to be built before the deadline to the minimum viable product. The remaining features can then be built as extensions to ensure there is always a workable product.
Budget	There is no budget for Art assets. This is an issue because the application is very abstract, and images would help to sell the concept.	Use creative commons licensed assets. This will allow images to be used for free as long as they are credited, and if budget becomes available down the line they can be swapped out in a patch.
Hardware / Software Platform	The user must be connected to the internet to interface with the system. This is an issue in areas of low network reliability, and some consumers may want to use the product offline.	The user must accept the constraint in the short term, however an extension could be planned to move the data to local storage if there is demand.
Persistent Storage & transactions	The inventory system will not be connected to the user's financial ledger or customer details. This means that there will be manual work for the user to make sure that and additional bookkeeping is aligned.	The user must keep any additional records separately. However an extension could be planned to add bookkeeping functionality to the system if there is demand.
Usability	The user may not be familiar with this type of system. This could lead to errors in data entry which would reduce the functional benefit of the app.	User training must take place to reduce the likelihood of user error.
Performance Requirements	Large numbers of records may be required to be kept by the application. This could cause slowdowns and reduce the app's functionality.	The back end should be designed to be as extensible as possible to ensure large volumes are handled efficiently.

**Description here**

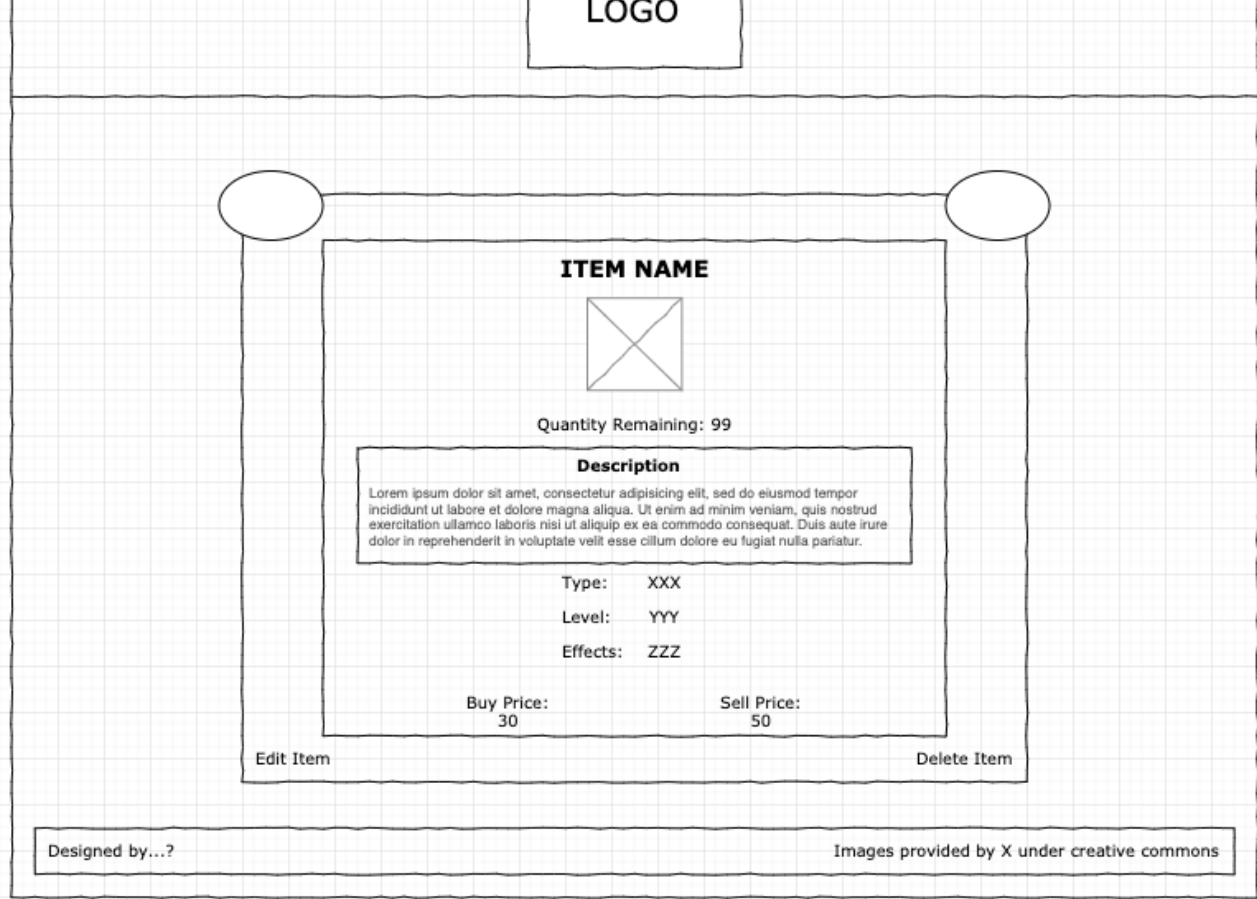
The implementation constraints plan developed for this project is shown above. The biggest concern here is the time constraint.

Unit	Ref	Evidence	
P	P.5	User Site Map	



**Paste Screenshot here**  
**Description here**  
Above is the site map for the project.

Unit	Ref	Evidence	
P	P.6	2 Wireframe Diagrams	



WIREFRAME - INDEX PAGE



Paste Screenshot here

### Description here

These are the wireframes for the inventory index page and for the item details page. The final result was very close.

## Week 5

Unit	Ref	Evidence	
P	P.10	Example of Pseudocode used for a method	

### Paste Screenshot here

```
1  def usable_by()
2
3  get back from the database all Character Class records which match the
  * type id
4
5  for each of the records returned, create a new array of character class
  * objects
6
7  return the array
8
9  end;
10
```

### Description here

This pseudocode describes what the method **usable\_by** should be doing on the types object.

Using pseudocode was helpful in planning what steps the method needs to go through to accomplish the required result.

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	

## Create Item

Name: <input type="text" value="TestItem"/>	Level: <input type="text" value="1"/>
Description: <input type="text" value="This is a test item"/>	Effects: <input type="text" value="none"/>
Image Name: <input type="text"/>	Buy Price: <input type="text" value="6"/>
Type: <input type="text" value="Armour (Clothing)"/>	Sell Price: <input type="text" value="12"/>
Quantity: <input type="text" value="3"/>	

Paste Screenshot here

Description here

The above screenshots show the application screen that allows the user to create a new item. The first screenshot is the data entry form, and the second screenshot shows the resulting page which retrieves the stored data entered above.

## TestItem



Description: This is a test item

Type: Armour	Buy Price: 6.0
Quantity: 3	Sell Price: 12.0
Level: 1	Margin: 6.0
Effects: none	

Sell Item:	1	Sell
Buy Item:	1	Buy

Edit Item
Delete Item



Unit	Ref	Evidence	
P	P.14	Show an interaction with data persistence. Take a screenshot of: <ul style="list-style-type: none"> <li>* Data being inputted into your program</li> <li>* Confirmation of the data being saved</li> </ul>	

Paste Screenshot here

Type List			
<a href="#">Add New Type</a>			
Name: TestType Category: Shield Usable By: <a href="#">Details</a>	Name: Armour Category: Clothing Usable By: Warrior Ranger <a href="#">Details</a>	Name: Shoes Category: Clothing Usable By: Warrior Mage Ranger <a href="#">Details</a>	Name: Necklace Category: Jewellery Usable By: Mage <a href="#">Details</a>

Description here

In the above screenshots, you can see the data being input to create a new item type. In the second screenshot, you can see that the entered item type has become persistent, and can be used to create new items.

Unit	Ref	Evidence	
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: <ul style="list-style-type: none"> <li>* The user requesting information or an action to be performed</li> <li>* The user request being processed correctly and demonstrated in the program</li> </ul>	

Paste Screenshot here

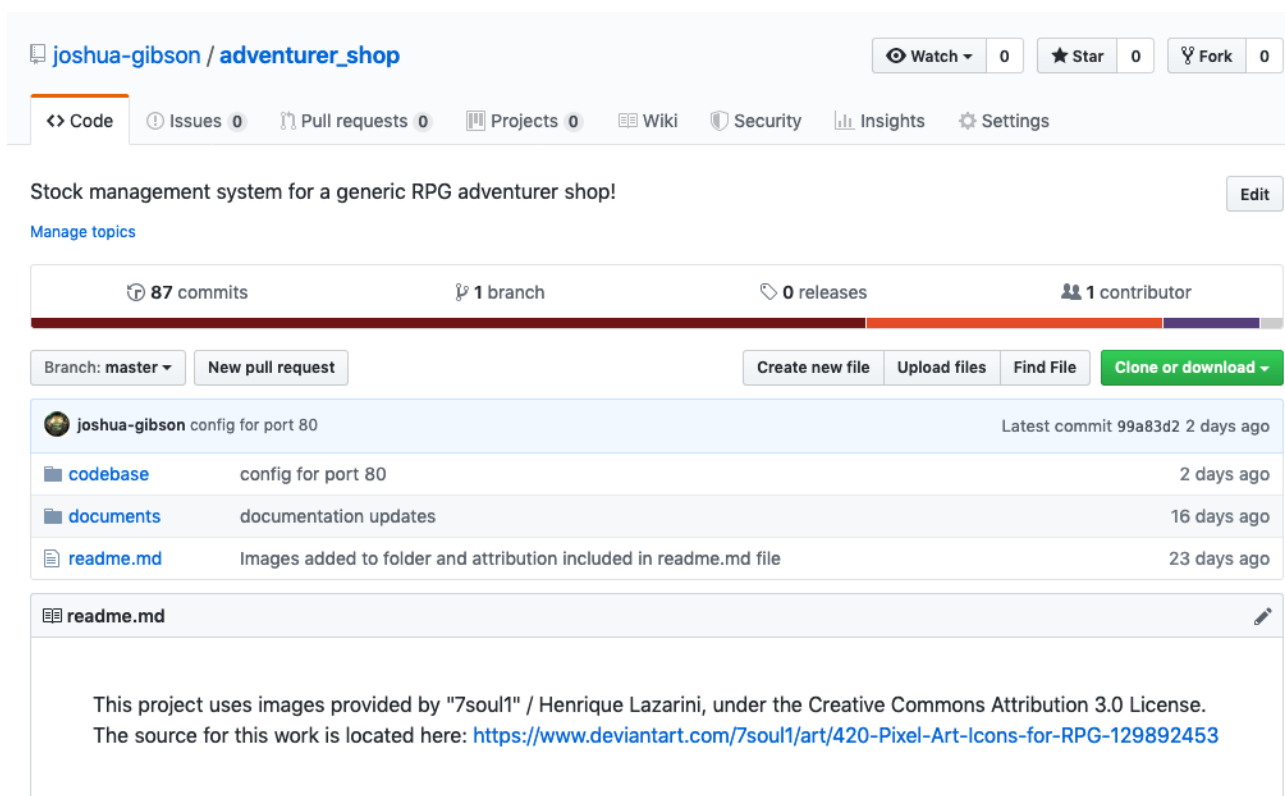


---

**Description here**

It the first screenshot, you can see in the 'sell item' square, the user has entered input that they wish to sell 2 of the selected items.  
In the second screenshot, after the user has pressed the sell button, you can see that the quantity has reduced by 2.

Unit	Ref	Evidence	
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.	



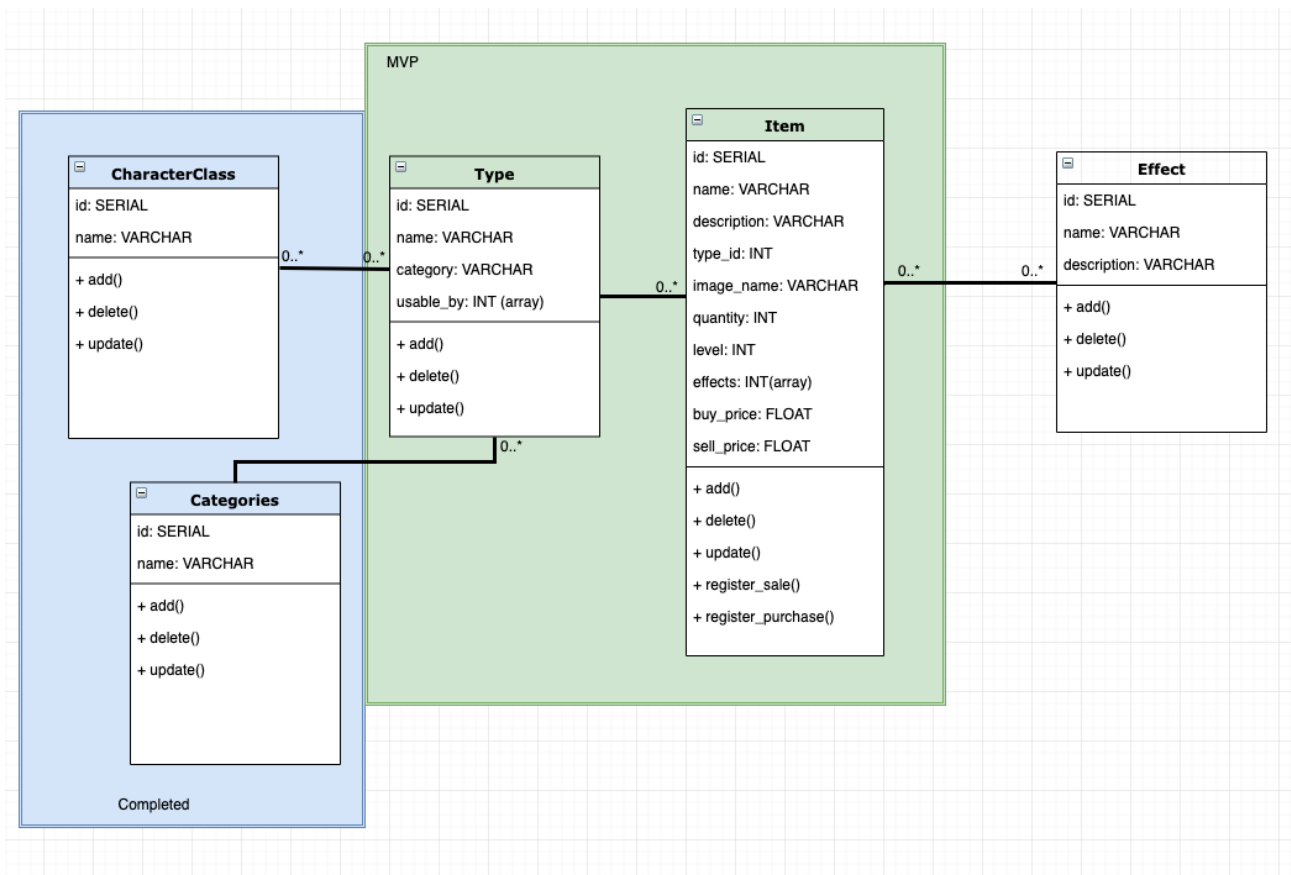
**Paste Screenshot here**

**Description here**

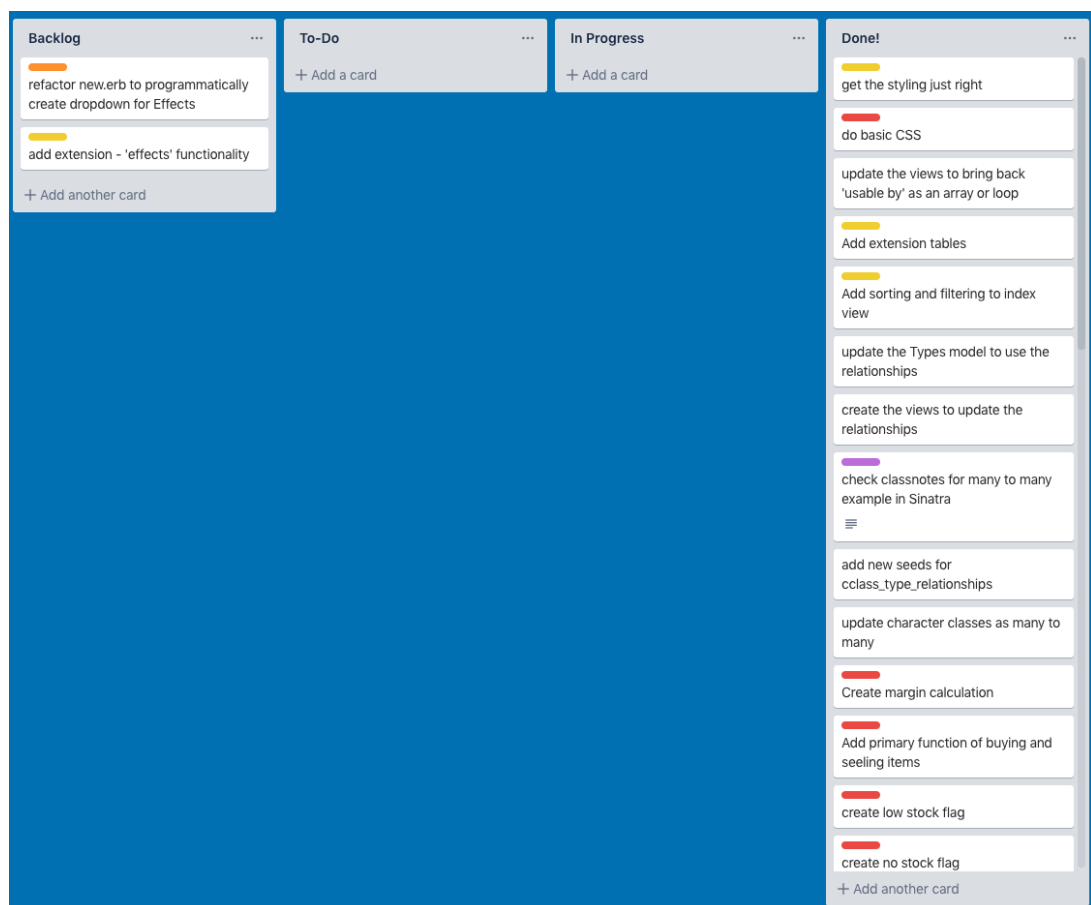
The above screenshot shows the Github repo for my solo project. This can be viewed in the browser via the following URL:

[https://github.com/joshua-gibson/adventurer\\_shop](https://github.com/joshua-gibson/adventurer_shop)

Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.



Paste Screenshot here



### **Description here**

The screenshots above shows how the project was planned and how that changed over time. In the first screenshot, you can see in the green border the 'MVP', or minimum viable product. This was completed about half way through the project. At that stage the classes in the blue border were added as an extension.

The second screenshot shows the Trello board which was used in the planning process. As tasks were completed, these moved from the Backlog column through to the Done column.

### **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	

### **Paste Screenshot here**

```
fetch('https://ghibliapi.herokuapp.com/films/')
  .then((response) => response.json())
  // .then(fdata => this.films = fdata)
  .then(fdata => this.createPosterArray(fdata))
  .then(posData => setTimeout(() => this.addPosterToFilm(), 500));
```

### Description here

The first screenshot shows the API call to the Studio Ghibli API (<https://ghibliapi.herokuapp.com/>). This API allows access to information about Studio Ghibli films. The second screenshot shows my app which makes use of this information to display it to the user based on their inputs.

# Studio Ghibli Films

Select Film:

Whisper of the Heart

## Whisper of the Heart



Shizuku lives a simple life, dominated by her love for stories and writing. One day she notices that all the library books she has have been previously checked out by the same person: 'Seiji Amasawa'. Curious as to who he is, Shizuku meets a boy her age whom she finds infuriating, but discovers to her shock that he is her 'Prince of Books'. As she grows closer to him, she realises that he merely read all those books to bring himself closer to her. The boy Seiji aspires to be a violin maker in Italy, and it is his dreams that make Shizuku realise that she has no clear path for her life. Knowing that her strength lies in writing, she tests her talents by writing a story about Baron, a cat statuette belonging to Seiji's grandfather.

Release Date: 1995  
 Producer: Toshio Suzuki  
 Director: Yoshifumi Kondō  
 RT Score: 91

People

Locations

Unit	Ref	Evidence	
P	P.2	Take a screenshot of the project brief from your group project.	

```
# Educational App

The BBC are looking to improve their online offering of educational content by developing some interactive browser applications that display information in a fun and interesting way. Your task is to make an a Minimum Viable Product or prototype to put forward to them – this may only be for a small set of information, and may only showcase some of the features to be included in the final app.

## MVP

A user should be able to:

- view some educational content on a particular topic
- be able to interact with the page to move through different sections of content

## Example Extensions

- Use an API to bring in content or a database to store information.
- Use charts or maps to display your information to the page.

## API, Libraries, Resources

- https://www.highcharts.com/ HighCharts is an open-source library for rendering responsive charts.
- https://leafletjs.com/ Leaflet is an open-source library for rendering maps and map functionality.
```

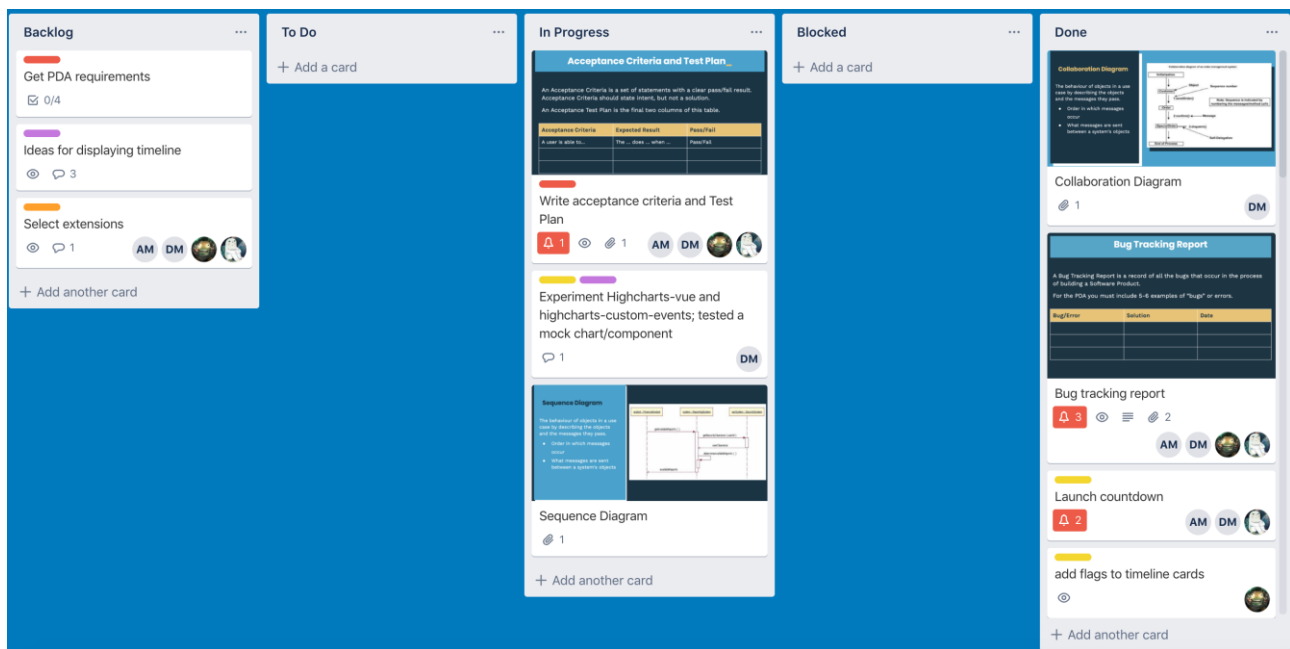
Paste Screenshot here

Description here

This is the project brief. Based on the remit, we decided to build an app that shows space launches, both historically and future launches via an existing API.

Unit	Ref	Evidence	
P	P.3	Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.	

Paste Screenshot here



### Description here

Above is the Trello board we used to manage our group project.

Unit	Ref	Evidence	
P	P.4	Write an acceptance criteria and test plan.	

	Acceptance Criteria and Test Plan		
	An Acceptance Criteria is a set of statements with a clear pass/fail result;		
	Acceptance Criteria should state intent, but not a solution;		
	An acceptance Test Plan is the final two columns of this table;		
	<b>Acceptance Criteria</b>	<b>Expected Result</b>	<b>Pass/Fail</b>
1	In LaunchTimeline.vue, component, through the Method bottomVisible () a user is able to get to the bottom of the page and then be able to load more stuff; to be able to do that we also wrote a constURL function in ApiService.js and further updated fetchData(offset) from LaunchTimeline.vue; we also added a watch: method in LaunchTimeline.vue	More items should be loaded as the user scrolls down to the bottom of the page;	Pass
2	In API Service we are writing a const function getDateStr in order to convert the date objects in string, so that we could display launches from a certain date to a certain date;		
3	We worked on adding a nav bar to our main page, above the LaunchSplash component	We expect the user to view a horizontal nav bar with few options displayed horizontally on the nav	Pass
4	We used Imsomnia as part of our testing plan; this helped us bring historic data about launches in our DB	Create own API with historic launches	Pass

### Paste Screenshot here

### Description here

Above is the acceptance criteria and test plan built by the group as part of the project.



## Week 9

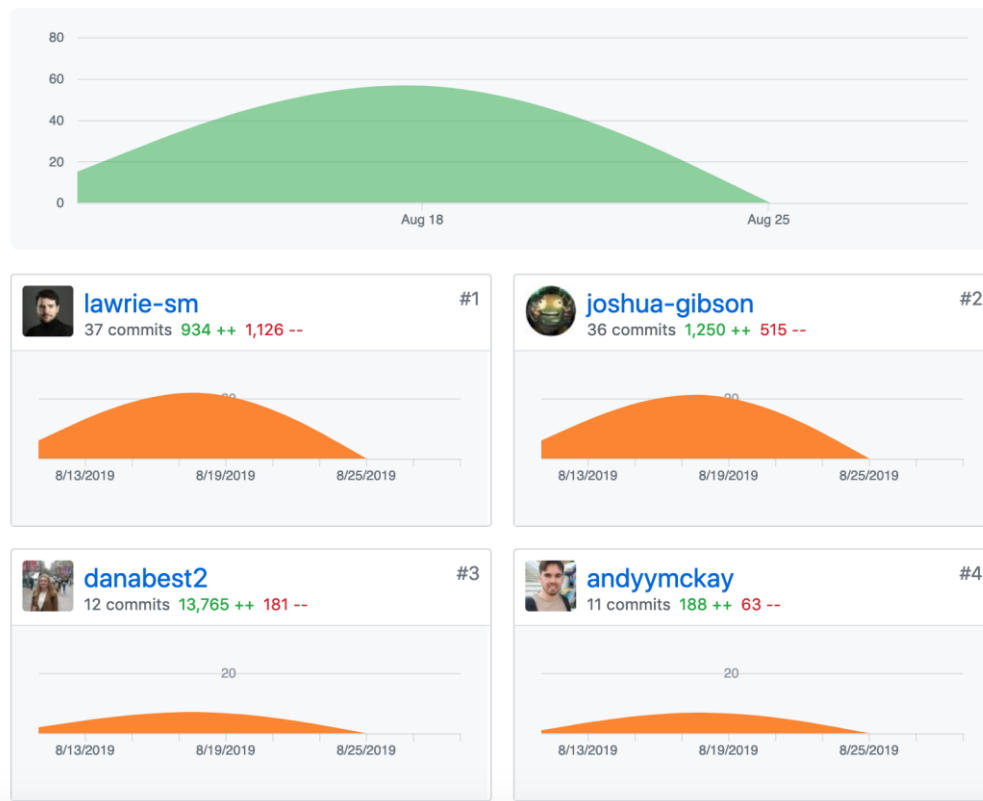
Unit	Ref	Evidence
P	P.1	Take a screenshot of the contributor's page on Github from your group project to show the team you worked with.

### Paste Screenshot here

Aug 11, 2019 – Aug 29, 2019

Contributions: Commits ▾

Contributions to master, excluding merge commits



### Description here

This is a screenshot of the contributors page for our project.

## Week 11

Unit	Ref	Evidence
P	P.18	<p>Demonstrate testing in your program. Take screenshots of:</p> <ul style="list-style-type: none"> <li>* Example of test code</li> <li>* The test code failing to pass</li> <li>* Example of the test code once errors have been corrected</li> <li>* The test code passing</li> </ul>

```

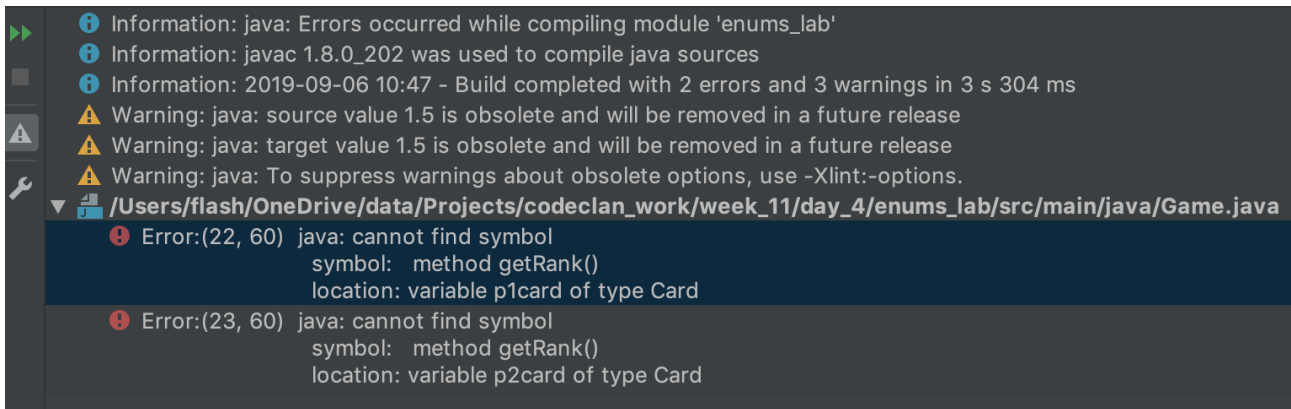
1  import org.junit.Before;
2  import org.junit.Test;
3
4  import static org.junit.Assert.assertEquals;
5
6  public class CardTest {
7
8      Card card;
9
10     @Before
11     public void before() { card = new Card(SuitType.HEARTS, RankType.QUEEN); }
12
13
14     @Test
15     public void canGetSuit() { assertEquals(SuitType.HEARTS, card.getSuit()); }
16
17
18     @Test
19     public void canGetRank() { assertEquals(RankType.QUEEN, card.getRank()); }
20
21
22     @Test
23     public void queenHasValue10(){
24         card = new Card(SuitType.HEARTS, RankType.QUEEN);
25         assertEquals( expected: 10, card.getValueFromEnum());
26     }
27
28 }

```

**Paste Screenshot here**

**Description here**

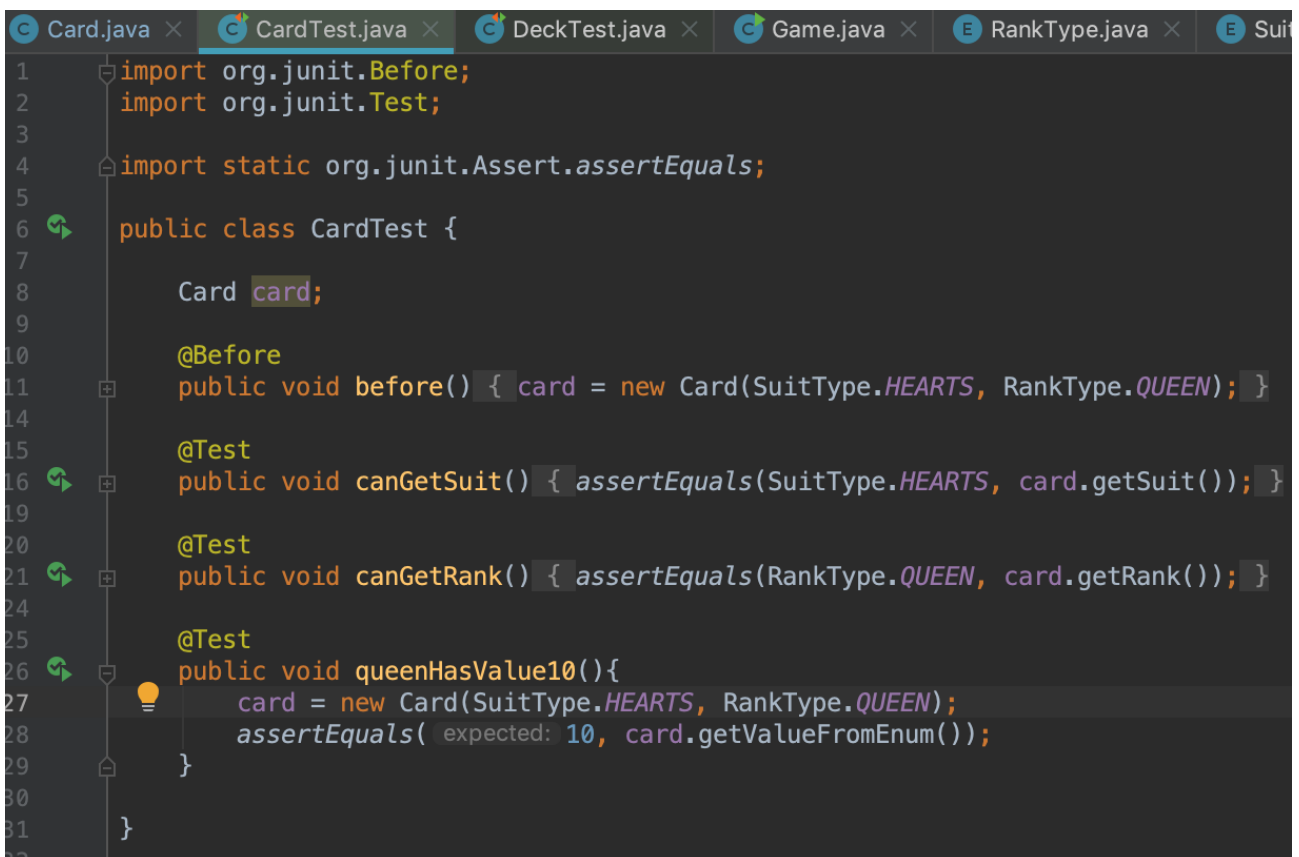
This shows the test code for the Card class. This will fail when run because the getRank() method hasn't yet been built.



**Paste Screenshot here**

**Description here**

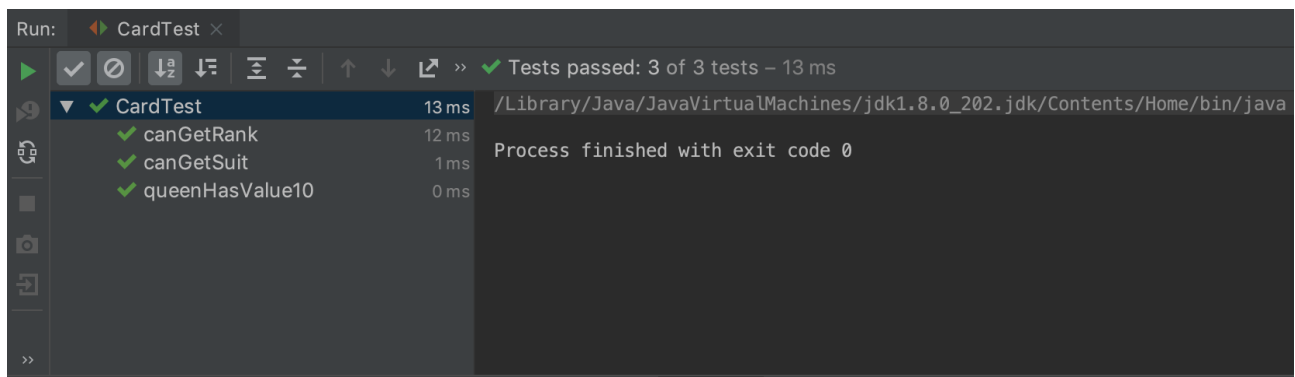
This shows the failure of the above tests, because of the missing method getRank().



**Paste Screenshot here**

**Description here**

This shows the test code with the errors corrected, the getRank() method has now been created.



**Paste Screenshot here**

**Description here**

This screenshot shows the tests running successfully after having been corrected.

Unit	Ref	Evidence
I&T	I.T.1	The use of Encapsulation in a program and what it is doing.

```
public class Card {

    private SuitType suit;
    private RankType rank;

    public Card(SuitType suit, RankType rank){
        this.suit = suit;
        this.rank = rank;
    }

    public SuitType getSuit() { return this.suit; }

    public RankType getRank() {
        return rank;
    }

    public int getValueFromEnum() { return this.rank.getValue(); }

}
```

**Paste Screenshot here**

**Description here**

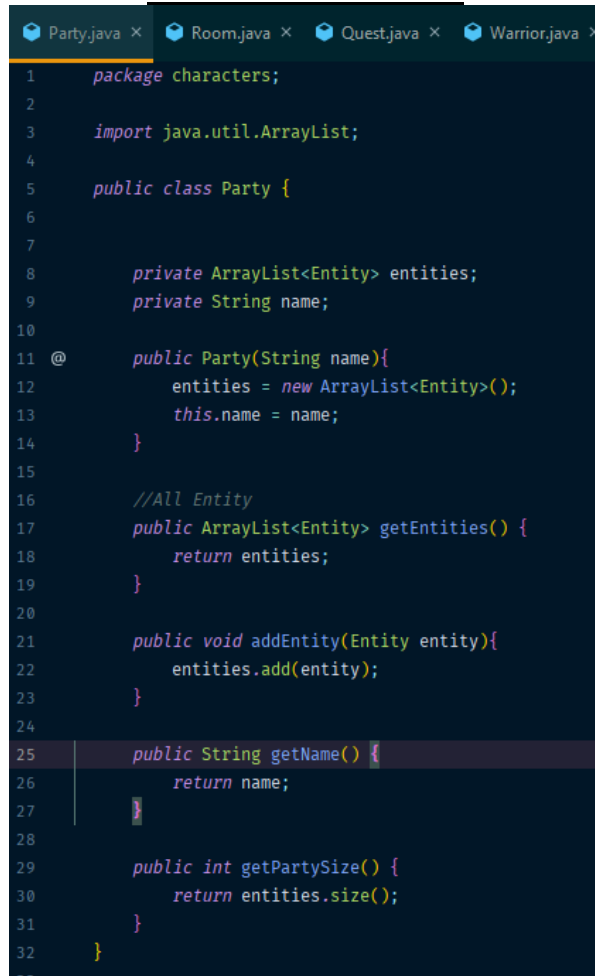
This screenshot is an example of encapsulation in the Card class. It has two properties, SuitType and RankType. These properties are set to 'private' at the top of the class so they cannot be accessed directly. The first two methods in the class are 'getters', their purpose is to allow the

properties to be accessed within the principles on encapsulation. They allow the properties to be read, but only as the programmer intended, and the values cannot be set directly.

## Week 12

Unit	Ref	Evidence	
I&T	I.T.7	The use of Polymorphism in a program and what it is doing.	

### Paste Screenshot here

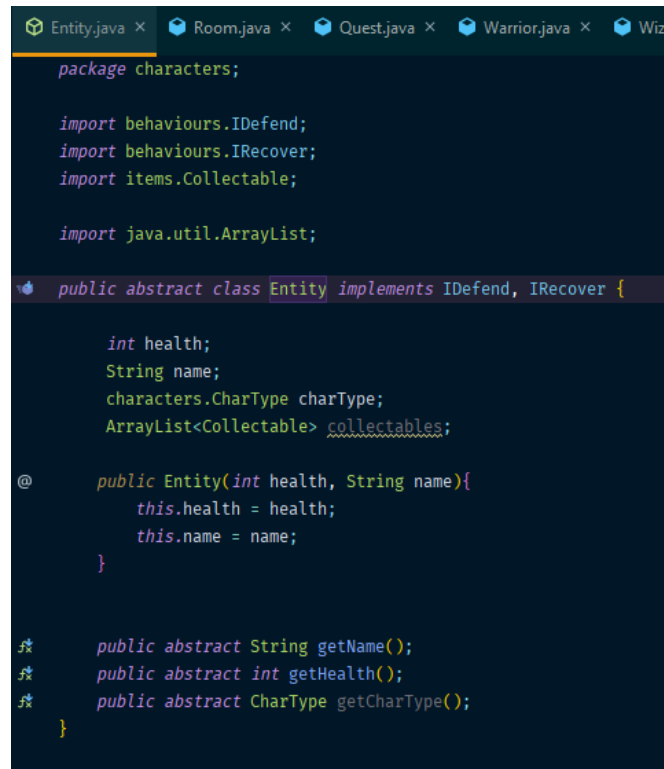


```
1 package characters;
2
3 import java.util.ArrayList;
4
5 public class Party {
6
7
8     private ArrayList<Entity> entities;
9     private String name;
10
11     @
12     public Party(String name){
13         entities = new ArrayList<Entity>();
14         this.name = name;
15     }
16
17     //All Entity
18     public ArrayList<Entity> getEntities() {
19         return entities;
20     }
21
22     public void addEntity(Entity entity){
23         entities.add(entity);
24     }
25
26     public String getName() {
27         return name;
28     }
29
30     public int getPartySize() {
31         return entities.size();
32     }
33 }
```

### Description here

The screenshot shows an object called Party, which includes an array list of Entities, which can be of different types (warrior, cleric, wizard). You can see also different methods which allow for the population of the array list, and retrieving information about it.

### Paste Screenshot here



```
Entity.java × Room.java × Quest.java × Warrior.java × Wiz
package characters;

import behaviours.IDefend;
import behaviours.IRecover;
import items.Collectable;

import java.util.ArrayList;

public abstract class Entity implements IDefend, IRecover {

    int health;
    String name;
    characters.CharType charType;
    ArrayList<Collectable> collectables;

    @ public Entity(int health, String name){
        this.health = health;
        this.name = name;
    }

    /* public abstract String getName();
    /* public abstract int getHealth();
    /* public abstract CharType getCharType();
}
```

### Description here

This shows the abstract class Entity, which is used in the arraylist on Party above. It implements the IDefend and IRecover interfaces, which allow the entity to defend from attach and recover health.

## Paste Screenshot here

```
Entity.java x Room.java x Quest.java x Warrior.java x Wizard.java x WarriorTest.java ... 2
package characters;

import behaviours.IAttack;
import behaviours.IDefend;
import items.Armour;
import items.Collectable;
import items.Weapon;

import java.util.ArrayList;

public class Warrior extends Entity implements IAttack {

    private ArrayList<Collectable> collectables;
    private Armour armour;
    private Weapon weapon;

    public Warrior(int health, String name, Armour armour, Weapon weapon){
        super(health, name);
        collectables = new ArrayList<Collectable>();
        this.armour = armour;
        this.weapon = weapon;
    }

    public ArrayList<Collectable> getCollectables() {
        return collectables;
    }

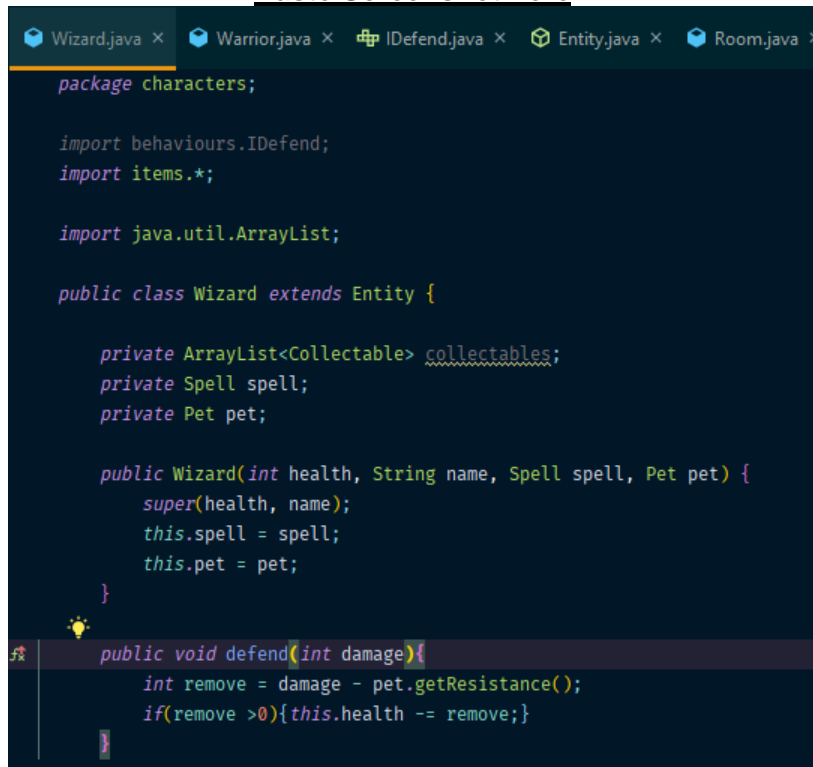
    public void defend(int damage){
        int remove = damage - armour.getResistance();
        if(remove > 0){this.health -= remove;}
    }

    public void attack(IDefend entity, int damage){
        Entity victim = (Entity) entity;
        entity.defend(damage);
        System.out.println(getName() + " attacks " + victim.getName() + " for " + damage + " ");
        if(victim.getClass().toString().toUpperCase().contains("WARRIOR")){
            Warrior vWarrior = (Warrior) victim;
            System.out.println(vWarrior.getName()+"'s armour mitigates " + vWarrior.getArmour().getResistance());
        }
        if(victim.getClass().toString().toUpperCase().contains("WIZARD")){
            Wizard vWizard = (Wizard) victim;
            System.out.println(vWizard.getName()+"'s pet mitigates " + vWizard.getPet().getHealth());
        }
        System.out.println(victim.getName() + " has " + victim.getHealth() + " health remain");
    }
}
```

## Description here

This is the Warrior class, which extends Entity. Here you can see that it implements the IDefend interface specified in the abstract class.

### Paste Screenshot here



```
package characters;

import behaviours.IDefend;
import items.*;

import java.util.ArrayList;

public class Wizard extends Entity {

    private ArrayList<Collectable> collectables;
    private Spell spell;
    private Pet pet;

    public Wizard(int health, String name, Spell spell, Pet pet) {
        super(health, name);
        this.spell = spell;
        this.pet = pet;
    }

    public void defend(int damage){
        int remove = damage - pet.getResistance();
        if(remove > 0){this.health -= remove;}
    }
}
```

### Description here

This is the Wizard class, similar to the Warrior class above, which is also extended from Entity and implements IDefend.

### Paste Screenshot here



```
package behaviours;

public interface IDefend {
    void defend(int damage);
}
```

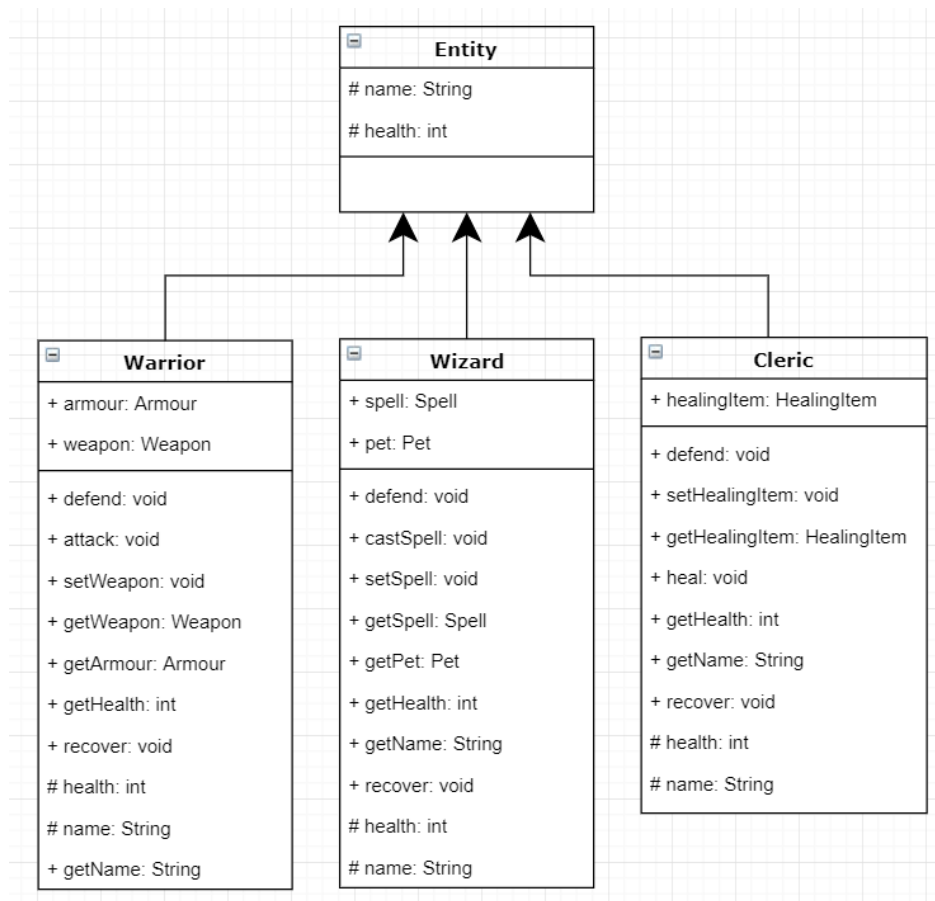
### Description here

This is the IDefend interface code referenced above.



Unit	Ref	Evidence	
A&D	A.D.5	An Inheritance Diagram	

**Paste Screenshot here**



**Description here**

Above is the inheritance diagram for the Fantasy Game lab, in which an entity was created as an abstract class and the Warrior, Wizard, and Cleric classes were extended from it.

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.	

### Paste Screenshot here

```
1 package characters;
2
3 import behaviours.IDefend;
4 import behaviours.IRecover;
5 import items.Collectable;
6
7 import java.util.ArrayList;
8
9 public abstract class Entity implements IDefend, IRecover {
10
11     int health;
12     String name;
13     ArrayList<Collectable> collectables;
14
15     public Entity(int health, String name){
16         this.health = health;
17         this.name = name;
18     }
19 }
```

### Description here

This screenshot shows the abstract class Entity.

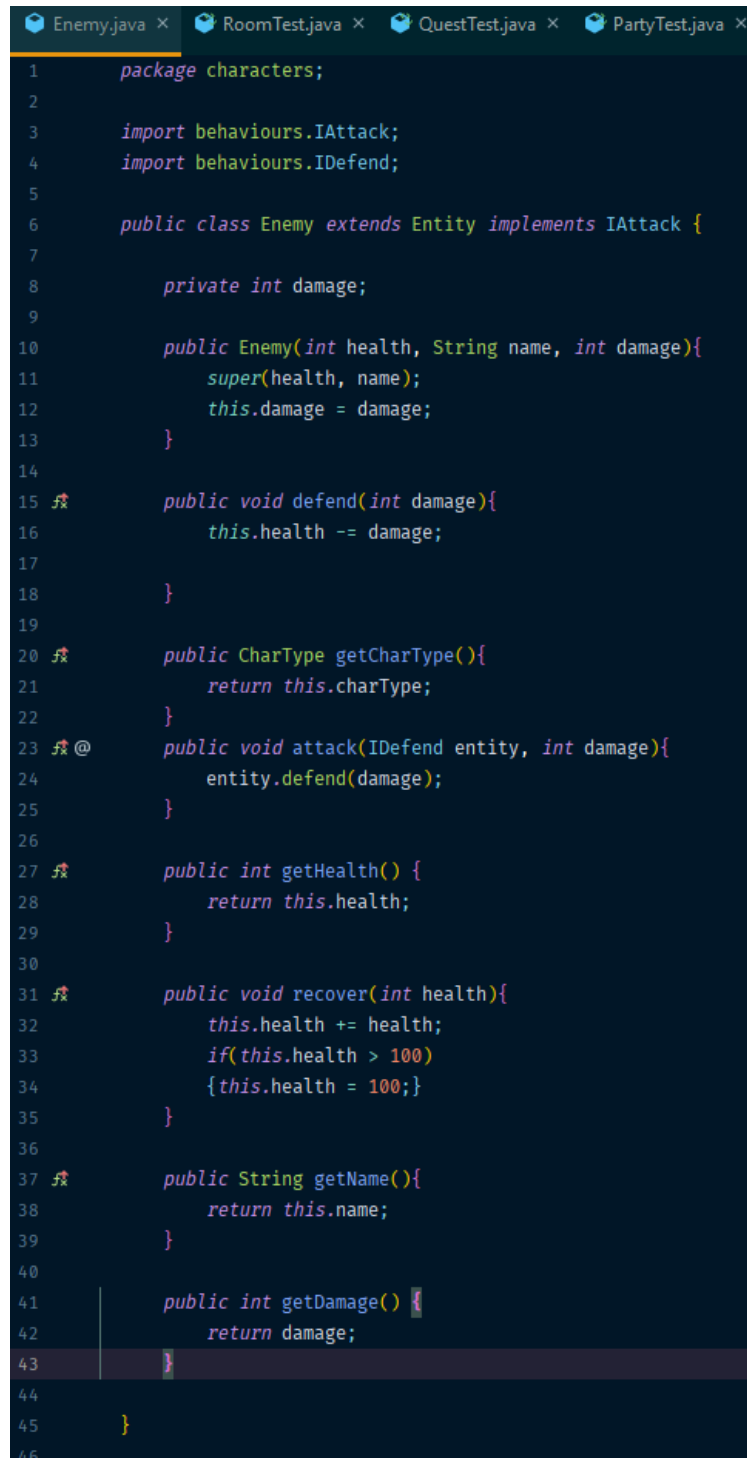
### Paste Screenshot here

```
1 package characters;
2
3 import behaviours.IAttack;
4
5 public class Enemy extends Entity implements IAttack {
6
7     private int damage;
8
9     public Enemy(int health, String name, int damage){
10         super(health, name);
11         this.damage = damage;
12     }
13 }
```

### Description here

This screenshot shows the class Enemy, which extends the abstract class Entity.

## Paste Screenshot here



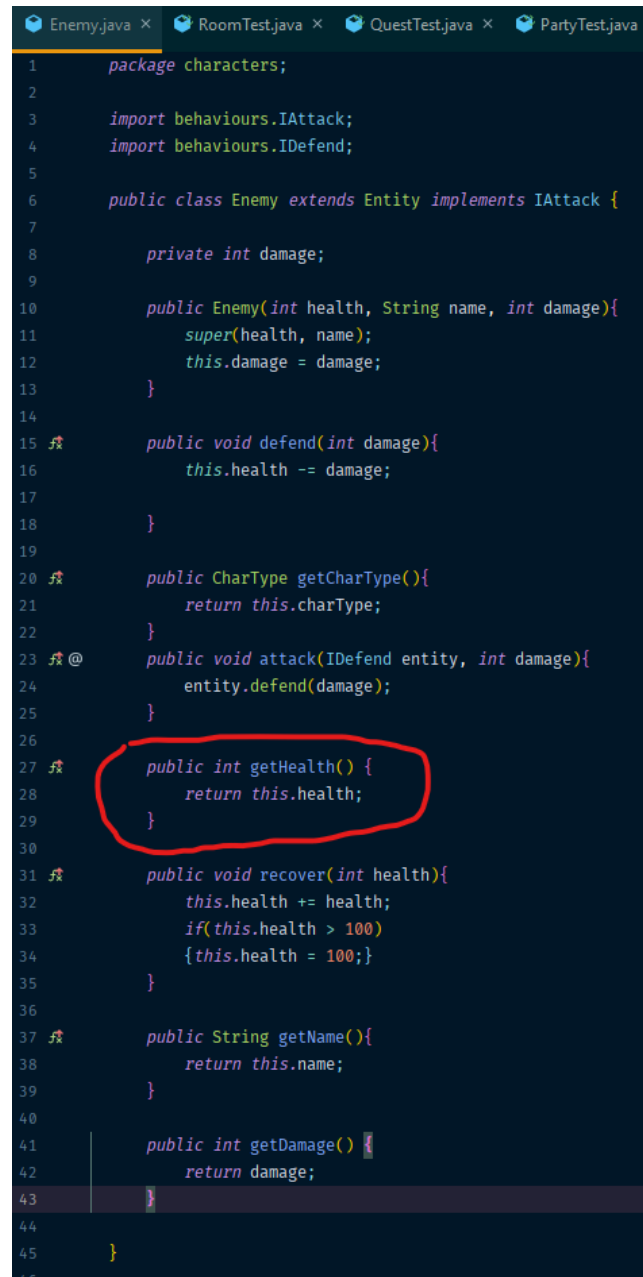
```
1 package characters;
2
3 import behaviours.IAttack;
4 import behaviours.IDefend;
5
6 public class Enemy extends Entity implements IAttack {
7
8     private int damage;
9
10    public Enemy(int health, String name, int damage){
11        super(health, name);
12        this.damage = damage;
13    }
14
15    public void defend(int damage){
16        this.health -= damage;
17    }
18
19
20    public CharType getCharType(){
21        return this.charType;
22    }
23    public void attack(IDefend entity, int damage){
24        entity.defend(damage);
25    }
26
27    public int getHealth() {
28        return this.health;
29    }
30
31    public void recover(int health){
32        this.health += health;
33        if(this.health > 100)
34            {this.health = 100;}
35    }
36
37    public String getName(){
38        return this.name;
39    }
40
41    public int getDamage() {
42        return damage;
43    }
44
45 }
```

```
public class EnemyTest {  
  
    Enemy enemy;  
    Warrior warrior;  
  
    @Before  
    public void SetUp(){  
        enemy = new Enemy(health: 100, name: "Gavin the Goblin", damage: 30);  
        warrior = new Warrior(health: 100, name: "Louise the Warrior", Armour.PLATE, Weapon.BOW);  
    }  
}
```

### Description here

These screenshots shows an object 'enemy' (below), created from the inherited class 'Enemy' (above).

### Paste Screenshot here



```
1 package characters;
2
3 import behaviours.IAttack;
4 import behaviours.IDefend;
5
6 public class Enemy extends Entity implements IAttack {
7
8     private int damage;
9
10    public Enemy(int health, String name, int damage){
11        super(health, name);
12        this.damage = damage;
13    }
14
15    public void defend(int damage){
16        this.health -= damage;
17    }
18
19
20    public CharType getCharType(){
21        return this.charType;
22    }
23    public void attack(IDefend entity, int damage){
24        entity.defend(damage);
25    }
26
27    public int getHealth() {
28        return this.health;
29    }
30
31    public void recover(int health){
32        this.health += health;
33        if(this.health > 100)
34            {this.health = 100;}
35    }
36
37    public String getName(){
38        return this.name;
39    }
40
41    public int getDamage() {
42        return damage;
43    }
44
45 }
```

### Description here

This method on the Enemy class returns 'health', which is inherited from the Entity class.

## Week 14

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.

### Paste Screenshot here

```
const filmNodes = this.props.films.map((film) => {
  return(
    <Film key = {film.id} name={film.name}>
      {film.url}
    </Film>
  );
});
```

---

### Description here

The algorithm above takes an ArrayList of films and for each creates a new container from the film's id, name, and URL. I've chosen this because I quite like the efficiency of the 'map' method.

### Paste Screenshot here

```
public void recover(int health){
  this.health += health;
  if(this.health > 100)
  {this.health = 100;}
}
```

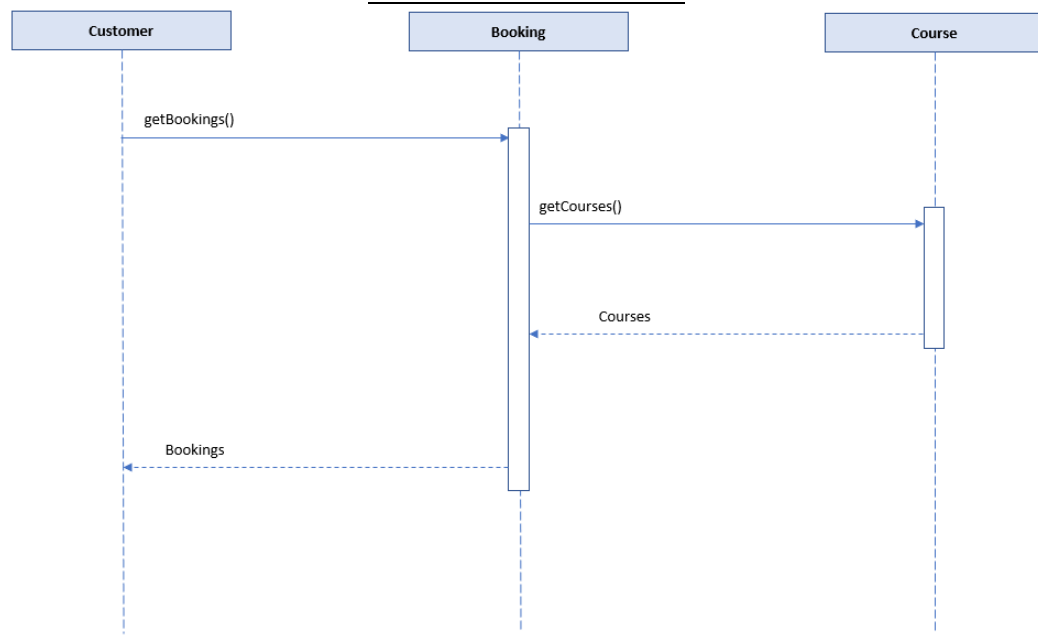
---

### Description here

The above algorithm is part of the adventure project, and it is triggered whenever a character recovers health. It takes in the amount of health to be recovered, and limits any recovery to cap the characters health at 100 points. I chose this because it is a good example of code that is very simple and reusable.

Unit	Ref	Evidence	
P	P.7	Produce two system interaction diagrams (sequence and/or collaboration diagrams).	

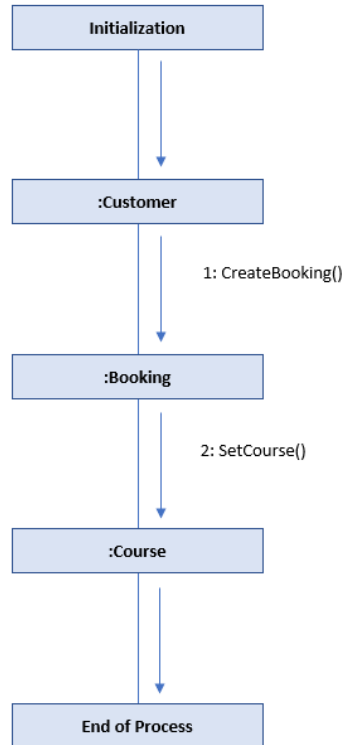
### Paste Screenshot here



### Description here

This screenshot shows a Sequence Diagram. You can see that when a Customer object invokes the getBookings method, the Booking object needs to retrieve the Course details from the Course object and return this information together to the Customer object.

### Paste Screenshot here

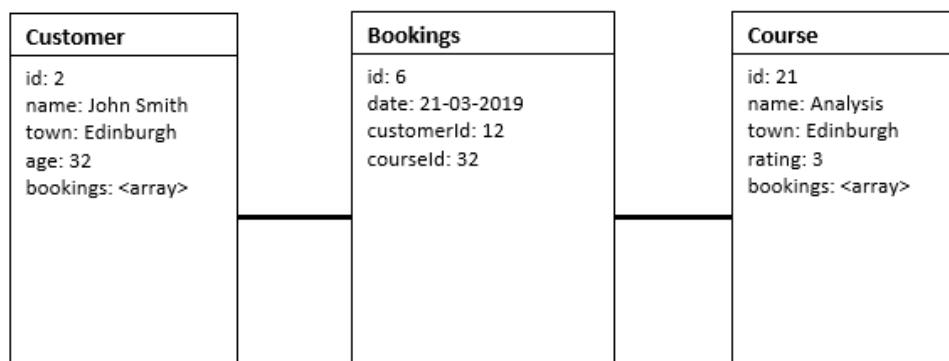


### Description here

This shows a Collaboration diagram, in which a customer can create a booking. The Creation request sends a message to the Booking object, which in turn needs to assign a course which is being booked, so sends a message to the Course object.

Unit	Ref	Evidence	
P	P.8	Produce two object diagrams.	

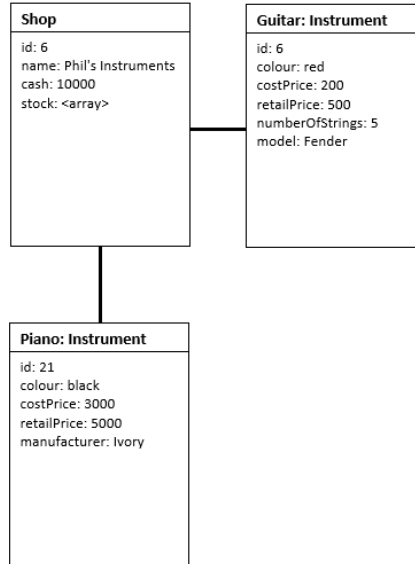
### Paste Screenshot here



### Description here

The above shows an object diagram of the course booking app. It shows that the Bookings table holds the relationship between the Customers and the Courses. Here we can see that the customer John Smith has booking ID 6, which means he is booked on the Analysis course on the 21<sup>st</sup> of March.

### Paste Screenshot here



### Description here

This shows an object diagram of an Instrument Shop app. This indicates that the Shop object holds an array of Stock, in which a Guitar and Piano are held, which are both Instruments.

Unit	Ref	Evidence	
P	P.17	Produce a bug tracking report	

### Paste Screenshot here



## Bug Tracking Report

Bug	Solution	Date
The RESTful route to POST a new dinosaur to the back end doesn't work.	The controller route had been overwritten. This code removed and added a new route "/tidy" to hold this instead.	01/10/2019
A dinosaur will only be added from the Lab page when clicking the add button twice.	The button was trying to do two things simultaneously - set the dino type and add the dino to the post. Therefore this needed to be in an "async" function before it would work.	02/10/2019
All the paddocks appearing in the top left of the paddock map	CSS naming of the paddocks had been renamed, reverted this and its working again.	29/09/2019
Receiving an error on the front end, "cannot map dinosaurs".	This was because the dinosaurs array was undefined. Fixed a problem with the fetch request and this is fixed.	30/09/2019
Dinosaur type not displaying on dinosaur cards	It seems that you cannot add new text to the cards in the way that has been done here. As we don't need the 'diet' text on the card, I've replaced this with the dino type and its all looking good now.	02/10/2019

### Description here

This is the bug tracking report for the Jurassic Park lab. It shows some of the issues encountered and how they were resolved.