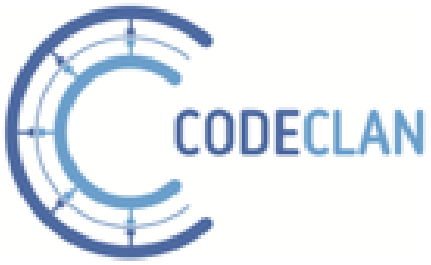


Evidence Gathering Document for SQA Level 8 Professional Developer Award.

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.

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	

Paste Screenshot here

```ruby
class Room

  attr_reader :number
  attr_accessor :songs, :seats

  def initialize(number)
    @number = number
    @songs = []
    @seats = []
  end

  def song_count
    @songs.count()
  end

  def add_guest_to_room(guest)
    @seats << guest
  end
```

```ruby
class RoomTest < MiniTest::Test

  def setup
    @room1 = Room.new(1)
    @song1 = Song.new("Hold up", "Beyonce")
    @song2 = Song.new("Hold up", "Beyonce")
    @song3 = Song.new("Hold up", "Beyonce")
    @guest1 = Guest.new("Tara", 25)
    @guest2 = Guest.new("Campbell", 70)
    @guest3 = Guest.new("Gregg", 56)
    @room1.songs.push(@song1, @song2)
  end
```

```ruby
def test_room_has_guests
  @room1.add_guest_to_room(@guest1)
  @room1.add_guest_to_room(@guest2)
  assert_equal(2, @room1.guests_count())
end
```

```
➜  homework_week2_day5 git:(master) ✗ ruby specs/room_spec.rb
Run options: --seed 5849

# Running:

.

Finished in 0.000984s, 1016.2602 runs/s, 1016.2602 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
➜  homework_week2_day5 git:(master) ✗
```

Description here

The first screenshot shows room which is set up to have an empty array of guests in its constructor and the method which will add guests to that array. The second shows the set up for a test before we add some guests to a room. The third shows the function in action and the last screenshot shows the test passing after we have counted that the guest objects have successfully been put in the room's guest array.

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	

Paste Screenshot here

```ruby
def whoIsFavouriteBuffyCharacter(someHash)
  for character in someHash[:characters]
    if character[:name] == "Spike"
      p character
    end
  end
end

characterHash = {
  characters: [{
  name: "Willow",
  age: "21",
  specialPower: "witch"},
  {name: "Buffy",
  age:"20",
specialPower: "slayer"},{
  name: "Spike",
  age: "121",
  specialPower: "vampire"
},{
  name: "angel",
  age: "156",
  specialPower: "good vampire"
}]
}

whoIsFavouriteBuffyCharacter(characterHash)
```

```
➜ Codeclan ruby hashPractice.rb
{:name=>"Spike", :age=>"121", :specialPower=>"vampire"}
➜ Codeclan
```

Description here

In the first screenshot we have a hash I made for Buffy the Vampire Slayer characters and a function which brings back the character Spike as the favourite character. In the second we have the result of running that function on the hash, which successfully brings back the favourite character as Spike.

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	

Paste Screenshot here

```ruby
def self.find(id)
  sql = "SELECT * FROM customers WHERE id = $1"
  values = [id]
  result = SqlRunner.run(sql, values)
  customer = self.new(result.first)
  return customer
end
```

```
codeclan_cinema=# select * from customers WHERE id = 5;
 id | name | funds
----+------+-------
  5 | Tara |    50
(1 row)

codeclan_cinema=#
```

Description here

In the first screenshot, you can see my method to find a customer by id and in the second you can see the result of that method successfully finding a customer by their id.

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	

Paste Screenshot here

```ruby
def albums()
  sql = "SELECT * FROM albums where artist_id = $1"
  values = [@id]
  results = SqlRunner.run(sql, values)
  return results.map {|album| Album.new(album)}
end
```

```
music_collection=# select * from artists;
 id |    name
----+------------
 30 | Sean Paul
 31 | Beenie Man
(2 rows)

music_collection=# select * from albums where artist_id = 30;
 id |         title          |   genre   | artist_id
----+------------------------+-----------+-----------
 27 | Dutty Rock             | dancehall |        30
 28 | King of the Dancehall  | dancehall |        30
(2 rows)

music_collection=#
```

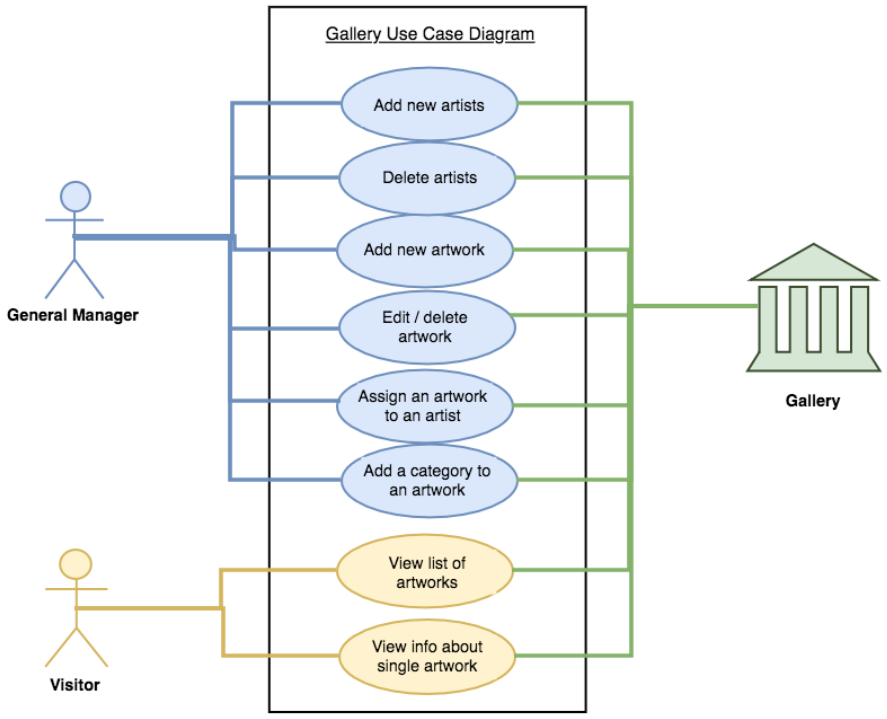
Description here

In the first screenshot, there is the method which sorts albums in this database by their artist id. In the second screenshot, you can see the result of this method successfully sorting albums by the specific artist given, i.e. Sean Paul.

Week 5

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

Paste Screenshot here

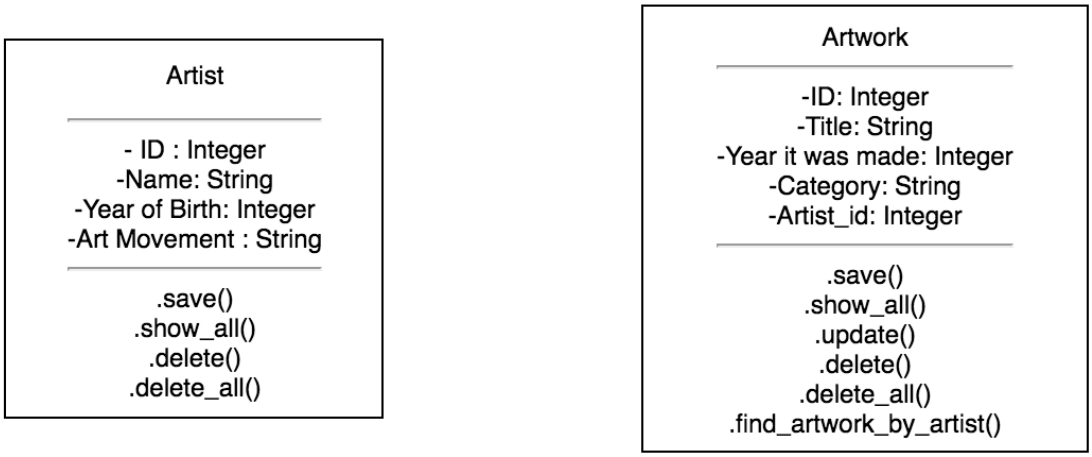


Description here

This is a use case diagram which demonstrates which functions are intended for which type of user. Here the general manager of my gallery web app has access to the bulk of the app's functionality with the visitor having some basic functions too.

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

Paste Screenshot here

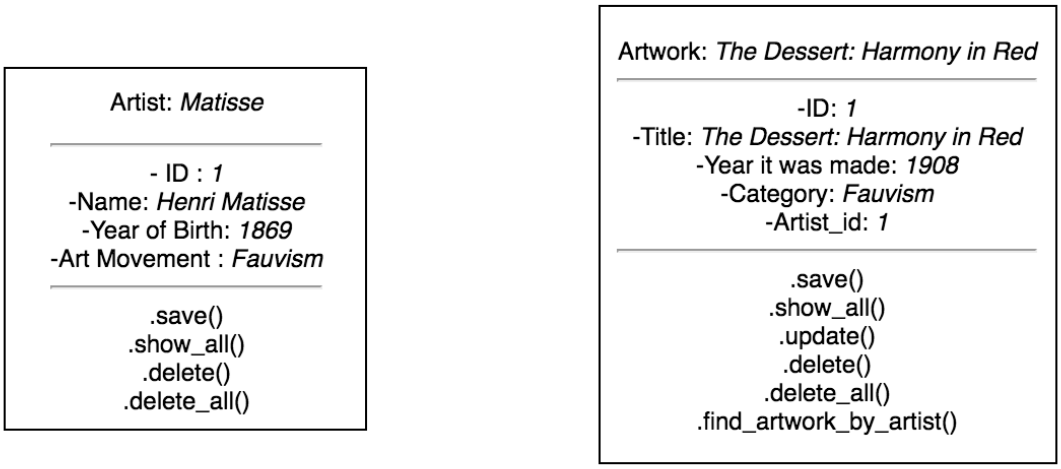


Description here

This is an example of a class diagram for a program which has two classes: one for artists and one for artworks. Their properties and methods are listed below.

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

Paste Screenshot here

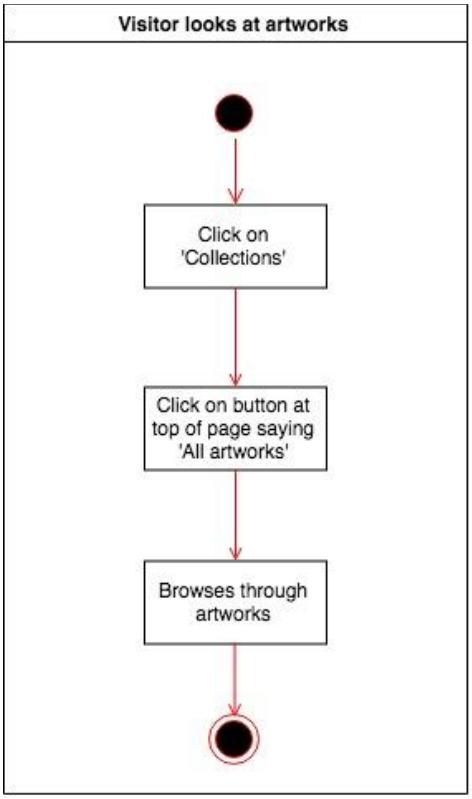


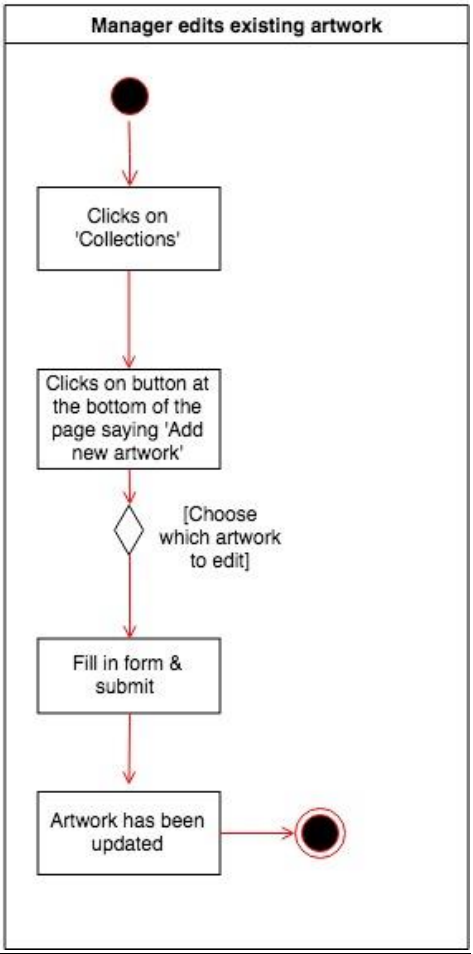
Description here

Here is an object diagram, which lays out examples of properties for each class of Artist and Artwork.

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

Paste Screenshot here





Description here

These are two activity diagrams for my gallery app, which show examples of interactions for the visitor and the manager. The visitor for example, can browse through the list of artworks a gallery has and the manager can add new artworks to the gallery's collection.

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

Implementation Constraints Plan : Sheet1

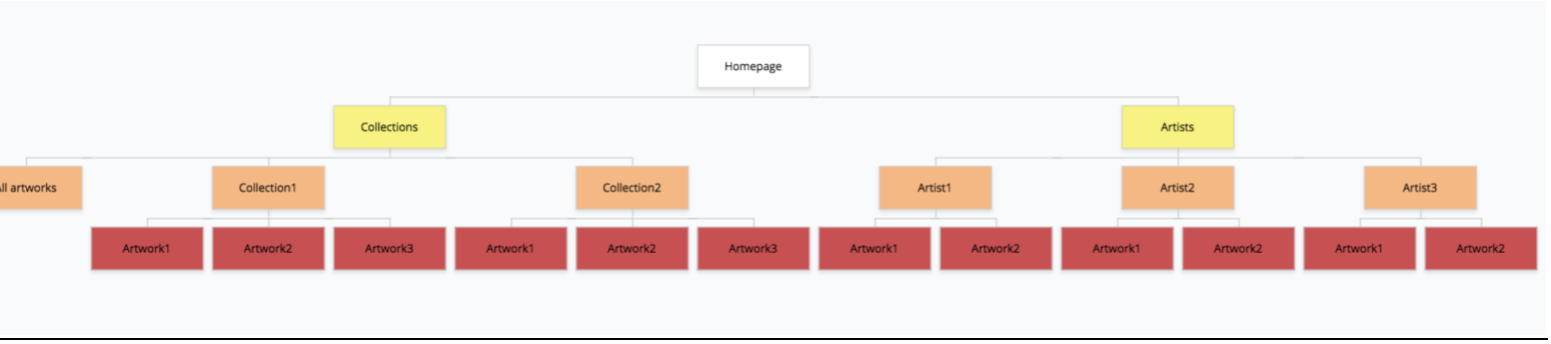
Topic	Possible Effect of Constraint on Product	Solution
Hardware and software platforms	If the function to filter artworks by category doesn't work.	Find out how to do this.
Performance requirement	If the page takes over 3 seconds to load.	Make sure all images are optimized for desktop and mobile use.
Persistent storage and transactions	The web app might not have enough memory to store all the images I would want for each artist	Only have a few images per artist.
Usability	The manager might want to upload a picture when they create a new artwork but I'm not 100% sure how to add this functionality yet	I will find out how to add this function to my 'add new artwork'
Budgets	If this were a real web app, the budget might not stretch to all the expensive visual elements I would want	This is not a real web app and there is only a week to create it, so this won't be an issue
Time limitations	Getting too distracted with the css and styling and not focussing enough on the functionality	Plan everything before actually writing any code, so that I wont be tempted to run away with the styling before I finish the back-end

Description here

Here is an implementation constraints plan I made for my project. Some of the criteria were more useful than others, for example budget wasn't really relevant this time and time limitations were particularly strict.

Unit	Ref	Evidence	
P	P.5	User Site Map	

Paste Screenshot here

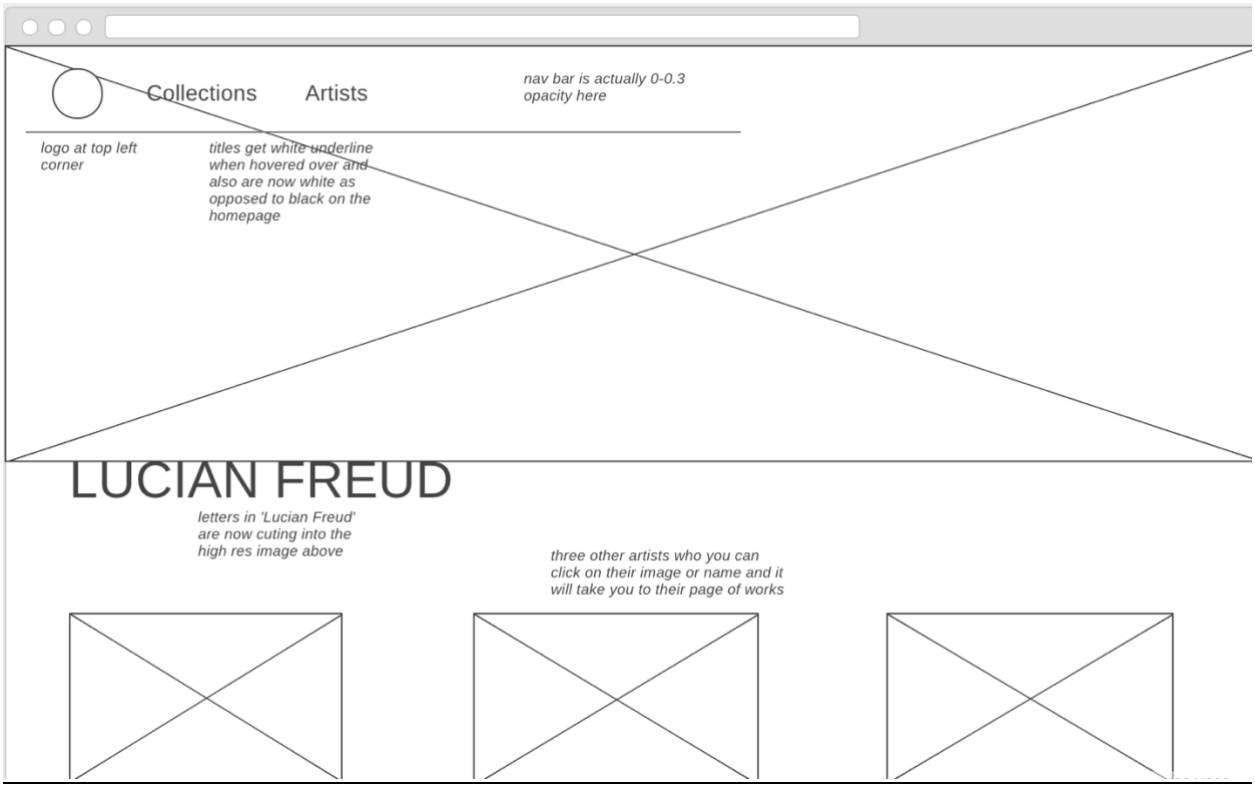


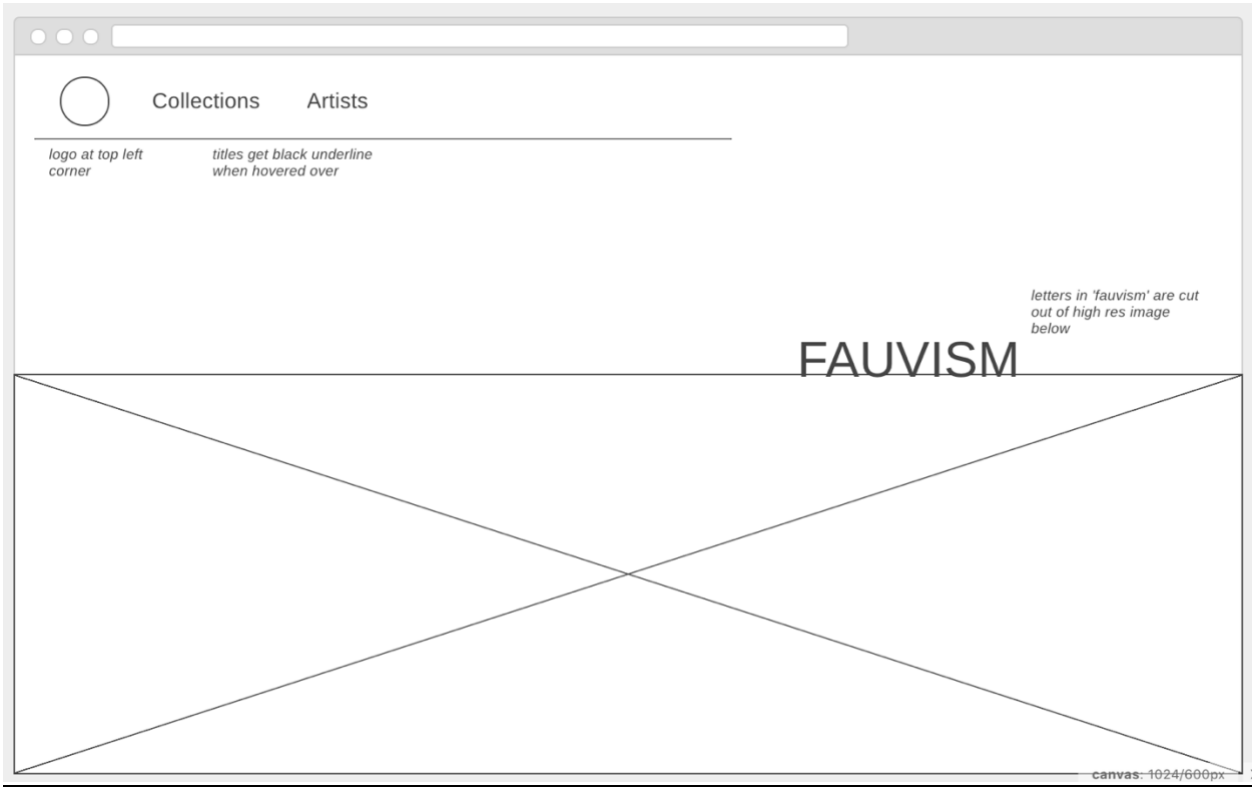
Description here

This is a sitemap of my art gallery web app. It has two main pages: collections and artists. You can visit a list of all artworks from the collections page and a list of all artists from the artists page. You can then further select individual artworks or individual artists from these pages.

Unit	Ref	Evidence	
P	P.6	2 Wireframe Diagrams	

Paste Screenshot here





Description here

These are two wireframes for my art gallery project. The first wireframe is for my 'Artists' page and the second is for the landing page for the website.

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

Paste Screenshot here

```
Customer.prototype.compareDifferenceInCollectionValue = function (customer) {
  //calculate the total value of each customer's collection
  let a = this.calculateCollectionTotalValue();
  let b = customer.calculateCollectionTotalValue();
  //check to see if the first customer's collection is of higher value than the second
  //if so, return the first customer object
    if (a > b){return this;}
    //then check if the second customer's collection is of higher value than the first
    //if so, return the second customer object
      else if (b > a) { return customer;}
      //if neither is of higher value than the other, they must be equal and so return the string
      //explaining both are the same value.
        return "Both are good";
};
```

Description here

In this method, I am trying to find the collection of the highest value. I pseudo coded each step of the process.

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	

Paste Screenshot here

shopBee

HOME NEW IN CLOTHES SHOES ACCESSORIES THE SHOP

Oval-frame gunmetal-tone sunglasses

Created in a gunmetal-tone colorway, this pair has a '90s-inspired design and mirrored oval-shaped lenses with full UV protection.

£125

EDIT ITEM

DELETE

shopBee

HOME NEW IN CLOTHES SHOES ACCESSORIES THE SHOP

Edit Product

Name : Change of name Description : Created in a gunmetal-t Category : Accessories Price : 20 Image : /images/Blue_sunglasse SAVE

© Tarheen 2018

shopBee

HOME NEW IN CLOTHES SHOES ACCESSORIES

Change of name
£20

Faye medium glossed-leather
£225

```
<p class="pageHeader">Edit Product</p>

<form action="/manager/accessories/$accessory.getId()/edit" method="post">

    <label for="title">Name :</label>
    <input class="inputBox" type="text" name="title" required="true" id="title" value="$accessory.getTitle()"/>

    <label for="description">Description :</label>
    <input class="inputBox" type="text" name="description" required="true" id="description" value="$accessory.getDescriptio

    <label for="category">Category :</label>
    <select id="category" name="category"> <!--for dropdown list-->
        #foreach($category in $categories)
        <option value="$category" #if ($category == $accessory.getCategory()) selected #end>$category.getDescription()</opt
        #end
    </select>


    <label for="price">Price :</label>
    <input class="inputBox" type="number" name="price" required="true" id="price" value="$accessory.getPrice()"/>

    <label for="image">Image :</label>
    <input class="inputBox" type="text" name="image" required="true" id="image" value="$accessory.getImage()"/>

    <input type="submit" value="Save" class="managerButton"/>

</form>
```

Description here

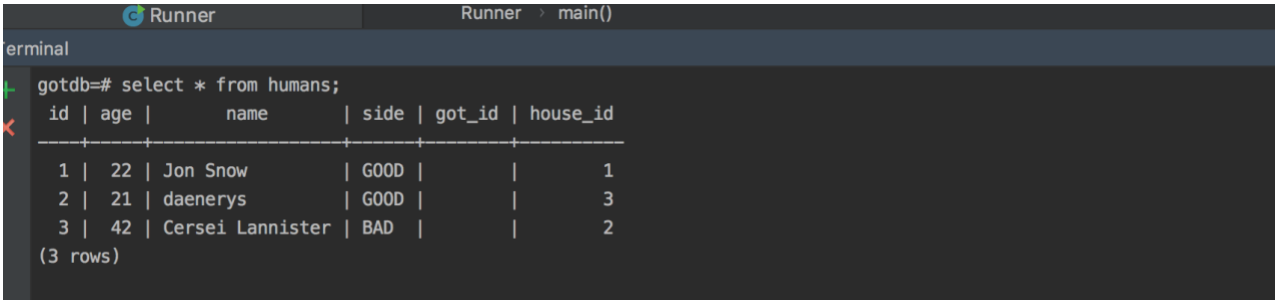
These are screenshots of my clothing store project. You can edit a product and save it to the database. Here I changed the name and price as you can see. Below is the code for the edit page template, specifically for accessories.

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	

Paste Screenshot here

```
Targaryen Daenerys = new Targaryen( name: "Daenerys Targaryen", age: 22, houseTargaryen, Side.UNKNOWN, mad: true);
DBHelper.saveOrUpdate(Daenerys);

Dragon Drogon = new Dragon(SpecialPower.FIREBREATHING, alive: true, strength: 500, attackPoints: 400, Daenerys);
DBHelper.saveOrUpdate(Drogon);

Dragon Rhaegal = new Dragon(SpecialPower.FIREBREATHING, alive: true, strength: 500, attackPoints: 400, Daenerys);
DBHelper.saveOrUpdate(Rhaegal);
```

```java
package db;

import ...

public class DBHelper {
    public static Session session;
    public static Transaction transaction;

    public static void saveOrUpdate(Object object){
        session = HibernateUtil.getSessionFactory().openSession();
        try {
            transaction = session.beginTransaction();
            session.saveOrUpdate(object);
            transaction.commit();
        } catch (HibernateException e) {
            transaction.rollback();
            e.printStackTrace();
        } finally {
            session.close();
        }
    }
```



Description here

This is an example of a function which saves data to a database and the result of that function successfully saving that data, as seen in terminal.

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	

Paste Screenshot here

shopBee

HOME NEW IN CLOTHES SHOES ACCESSORIES THE SHOP

New Product

Name : Dress Description : This lace gown has care Category : New In Price : 110 Image : 588/1062588_in_pp.jpg SAVE

© Tarheen 2018

shopBee

HOME NEW IN CLOTHES SHOES ACCESSORIES THE SHOP

Product added successfully.

© Tarheen 2018

shopBee

HOME NEW IN CLOTHES SHOES ACCESSORIES THE SHOP

Dress
£110

Leopard-print silk-chiffon blouse
£110

Gathered silk crepe de chine blouse
£80

```
<p class="pageHeader">New Product</p>

<form action="/add" method="post">

    <label for="title">Name :</label>
    <input class="inputBox" type="text" name="title" required="true" id="title"/>

    <label for="description">Description :</label>
    <input class="inputBox" type="text" name="description" required="true" id="description"/>

    <label for="category">Category :</label>
    <select id="category" name="category"> <!--for dropdown list-->
        #foreach($category in $categories)
        <option value="$category">$category.getDescription()</option>
        #end
    </select>

    <label for="price">Price :</label>
    <input class="inputBox" type="number" name="price" required="true" id="price"/>

    <label for="image">Image :</label>
    <input class="inputBox" type="text" name="image" required="true" id="image"/>

    <input type="submit" value="Save" class="managerButton"/>

</form>
```

```
get( path: "/addProduct", (res, req) -> {
    HashMap<String, Object> model = new HashMap<>();

    List<Category> categories = DBHelper.getAllCategories(); //for dropdownlist
    model.put("categories", categories);

    model.put("template", "templates/products/addProduct.vtl");
    return new ModelAndView(model,  viewName: "templates/layout.vtl");
}, new VelocityTemplateEngine());
```

```
post( path: "/add", (req, res) -> {
    String title = req.queryParams("title");
    String description = req.queryParams("description");
    Category category = Category.valueOf(req.queryParams("category"));
    int price = Integer.parseInt(req.queryParams("price"));
    String image = req.queryParams("image");
    List<Shop> shops = DBHelper.getAll(Shop.class);
    Shop shop = shops.get(0);

    Product product = new Product(title, description, category, price, image, shop);
    DBHelper.save(product);
    res.redirect( location: "/saved");
    return null;
}, new VelocityTemplateEngine());
```

Description here

The first few screenshots are of a user adding a new product to the clothing store. The last screenshots are of the 'add product template' and the get and post routes used to save the object to the database.

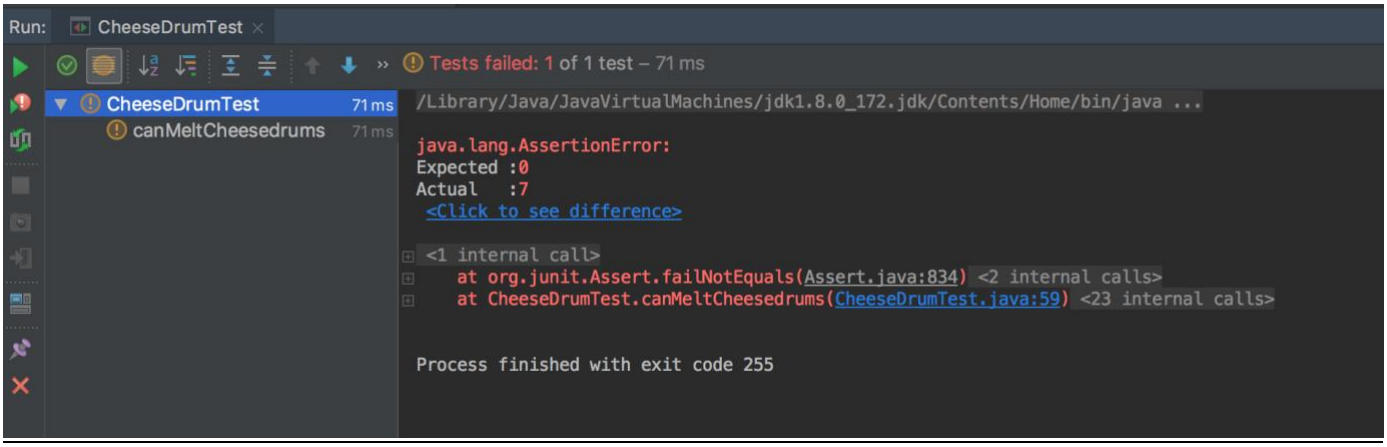
Unit	Ref	Evidence	
P	P.18	Demonstrate testing in your program. Take screenshots of: * Example of test code * The test code failing to pass * Example of the test code once errors have been corrected	

		* The test code passing

Paste Screenshot here

```
public  Cheesedrum(
String type, int priceBoughtFor, int priceSoldFor, String material, String colour, String sound)

{  super(type, priceBoughtFor, priceSoldFor, material, colour, sound);
    this.solidityLevel = 8;
}


public int getSolidity() {
    return this.solidityLevel;
}

public void bashHard(){
    if (this.solidityLevel >= 1)
    this.solidityLevel -= 1;
}
```

```
@Test
public void canMeltCheesedrums(){
    cheesedrum.bashHard();
    assertEquals( expected: 0, cheesedrum.getSolidity());

}
}
```



```
@Test
public void canMeltCheesedrums(){
    cheesedrum.bashHard();
    cheesedrum.bashHard();
    cheesedrum.bashHard();
    cheesedrum.bashHard();
    cheesedrum.bashHard();
    cheesedrum.bashHard();
    cheesedrum.bashHard();
    cheesedrum.bashHard();
    cheesedrum.bashHard();
    cheesedrum.bashHard();
    assertEquals( expected: 0, cheesedrum.getSolidity());
```

Description here

In these screenshots, you can see the Cheese drum class and a test to test the solidity of the cheese drums after they have had the 'bash hard' method applied to them. In the first test, it didn't work as the solidity as at 7 after one application of the 'bash hard' method, but the expected outcome was 0. Therefore, in the second test you needed to 'bash hard' 8 times for the drums to melt or get to 0.

Week 7

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

Paste Screenshot here

```java
package models;

import ...

@Entity
@Table(name ="NightKing")
public class NightKing extends Whitewalker{
private Set<Wight> wightArmy;

    public NightKing(){

    }

    public NightKing(SpecialPower specialPower, Boolean alive, int strength, int attackPoints, boolean areTheyBlue) {
        super(specialPower, alive, strength, attackPoints, areTheyBlue);
    }
    @OneToMany(mappedBy = "nightKing", fetch = FetchType.LAZY)
    public Set<Wight> getWightArmy() {
        return wightArmy;
    }

    public void setWightArmy(Set<Wight> wightArmy) {
        this.wightArmy = wightArmy;
    }
}
```

```java
package models;

import ...

@Entity
@Table(name = "whitewalkers")
public class Whitewalker extends MysticalCreature{
    private boolean areTheyBlue;
    private Set<IceDragon> iceDragons;

    public Whitewalker(){}

    public Whitewalker(SpecialPower specialPower, Boolean alive, int strength, int attackPoints, boolean areTheyBlue) {
        super(specialPower, alive, strength, attackPoints);
        this.areTheyBlue = areTheyBlue;
    }
    @Column(name="areTheyBlue")
    public boolean isAreTheyBlue() {
        return areTheyBlue;
    }
```

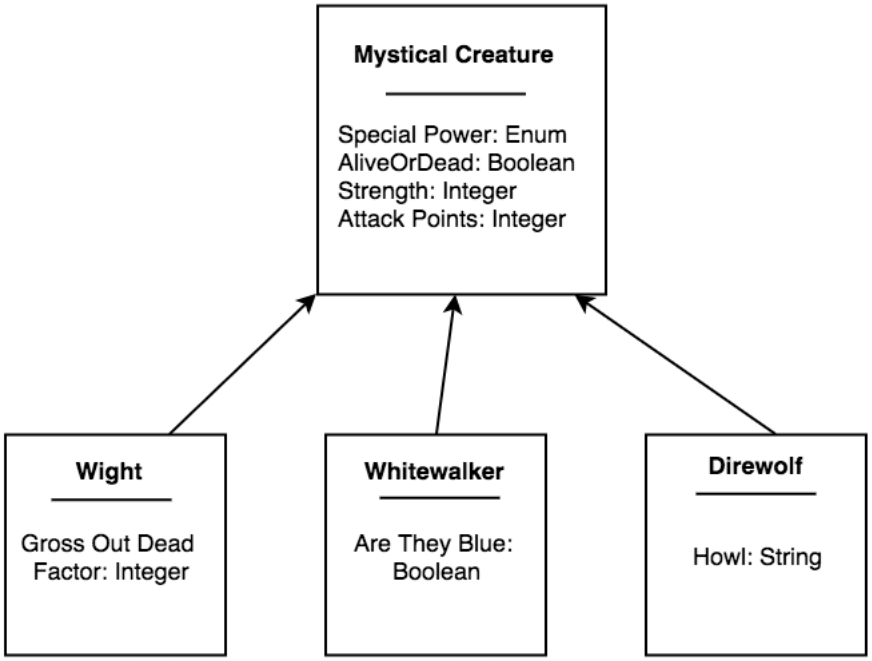
```java
package models;

import ...

@Entity
@Table(name ="mysticalCreatures")
@Inheritance(strategy = InheritanceType.JOINED)
public class MysticalCreature {
    private int id;
    private SpecialPower specialPower;
    private Boolean alive;
    private int strength;
    private int attackPoints;
    private GoTWorld goTWorld;


    public MysticalCreature(){

    }
```

<u>Description here</u>

These images demonstrate polymorphism in a program. They show that the Night King object is polymorphic as he extends from the Whitewalker class and the Mystical Creature class. He is therefore taking many forms as he takes properties from both class with multiple inheritance.

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

Paste Screenshot here



Description here

This is an inheritance diagram based on the creatures from Game of Thrones. All Wights, Whitewalkers and Direwolves inherit from the class of Mystical Creature, which has certain features. They each then have separate properties which differentiate them from each other, such as the Gross Out Dead Factor, whether they are blue or not and the fact that direwolves can howl.

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

Paste Screenshot here

```java
public class Cheesedrum extends Instrument {
    private int solidityLevel;

    public  Cheesedrum(
    String type, int priceBoughtFor, int priceSoldFor, String material, String colour, String sound)

    {  super(type, priceBoughtFor, priceSoldFor, material, colour, sound);
        this.solidityLevel = 8;
    }


    public int getSolidity() {
        return this.solidityLevel;
    }

    public void bashHard(){
        if (this.solidityLevel >= 1)
        this.solidityLevel -= 1;
    }
}
```

Description here

In this image, we can see the property of 'Solidity' of the Cheese Drum class being encapsulated. This means that this property is privatized, and you cannot access this property from outside the Cheese Drum class. To do this you would need to use the getter and setter methods.

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

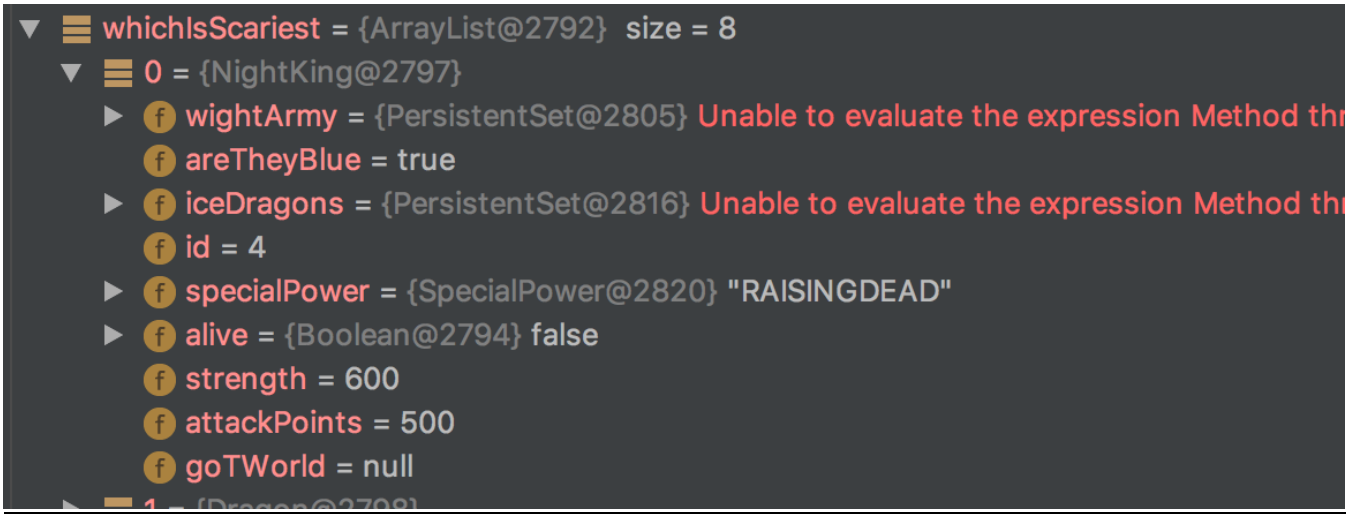
```java
package models;

import ...

@Entity
@Table(name ="mysticalCreatures")
@Inheritance(strategy = InheritanceType.JOINED)
public class MysticalCreature {
    private int id;
    private SpecialPower specialPower;
    private Boolean alive;
    private int strength;
    private int attackPoints;
    private GoTWorld goTWorld;


    public MysticalCreature(){

    }

    public MysticalCreature(SpecialPower specialPower, Boolean alive, int strength, int attackPoints) {
        this.specialPower = specialPower;
        this.alive = alive;
        this.strength = strength;
        this.attackPoints = attackPoints;
```

```
    }
    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    @Column(name ="id")
    public int getId() {
        return id;
    }

    @Enumerated(value = EnumType.STRING)
    public SpecialPower getSpecialPower() {
        return specialPower;
    }
    @Column(name ="alive")
    public Boolean getAlive() {
        return alive;
    }
    @Column(name ="strength")
    public int getStrength() {
        return strength;
    }
    @Column(name ="attackPoints")
    public int getAttackPoints() {
        return attackPoints;
    }
```

```
package models;

import ...

@Entity
@Table(name = "whitewalkers")
public class Whitewalker extends MysticalCreature{
    private boolean areTheyBlue;
    private Set<IceDragon> iceDragons;

    public Whitewalker(){}

    public Whitewalker(SpecialPower specialPower, Boolean alive, int strength, int attackPoints, boolean areTheyBlue) {
        super(specialPower, alive, strength, attackPoints);
        this.areTheyBlue = areTheyBlue;
    }
    @Column(name="areTheyBlue")
    public boolean isAreTheyBlue() {
        return areTheyBlue;
    }

    public void setAreTheyBlue(boolean areTheyBlue) {
        this.areTheyBlue = areTheyBlue;
    }
    @OneToMany(mappedBy = "whitewalker", fetch = FetchType.LAZY)
    public Set<IceDragon> getIceDragons() {
        return iceDragons;
    }
```

```java
package db;

import ...

public class DBMysticalCreature {
    private static Session session;
    private static Transaction transaction;

    public static List<MysticalCreature> orderByScariness(){
        session = HibernateUtil.getSessionFactory().openSession();
        List<MysticalCreature> scaryThings = null;
        try {
            Criteria cr = session.createCriteria(MysticalCreature.class);
            cr.addOrder(Order.desc("attackPoints"));
            scaryThings = cr.list();
        } catch (HibernateException e) {
            e.printStackTrace();
        } finally {
            session.close();
        }
        return scaryThings;
    }
```

```java
List<MysticalCreature> whichIsScariest = DBMysticalCreature.orderByScariness();
```



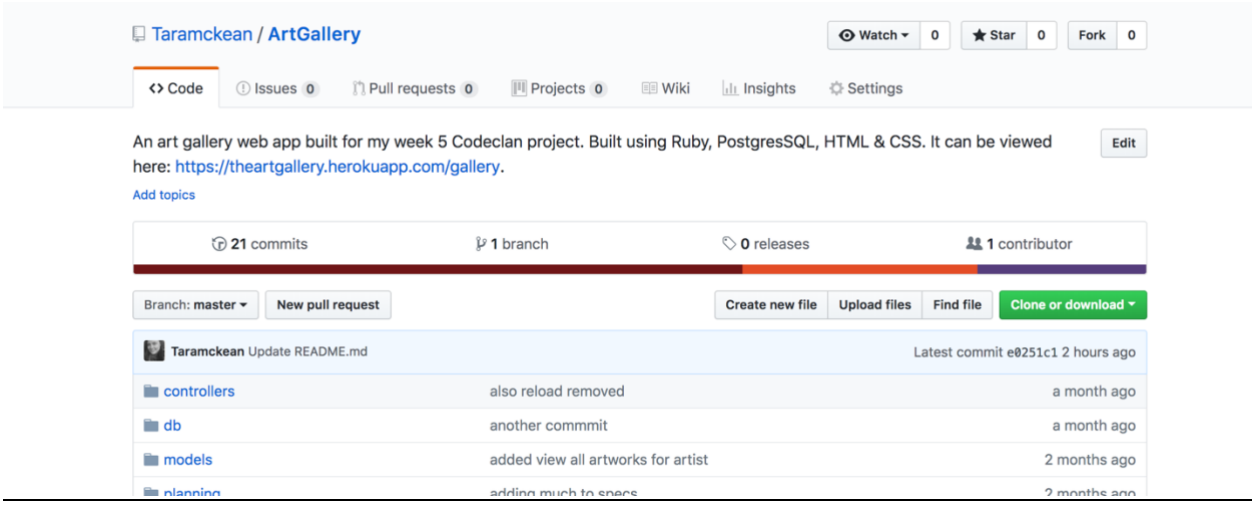
Description here

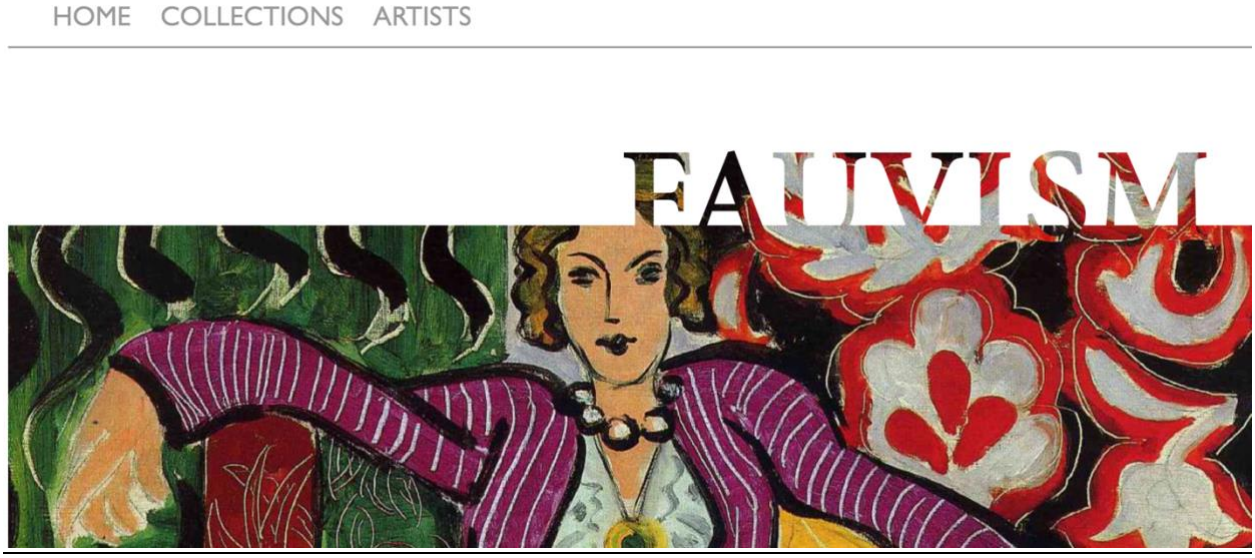
The first screenshot is of the Mystical Creature class and the second screenshot is of the Whitewalker class, which inherits from the Mystical Creature class. We then have a method which sorts mystical creatures in order of their attack points or 'scariness level'. The last two shots are of the method being used and the result of the method which returns the 'Night King', who extends Whitewalker and therefore Mystical Creature.

Week 10

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





[My project link: https://github.com/Taramckean/ArtGallery]

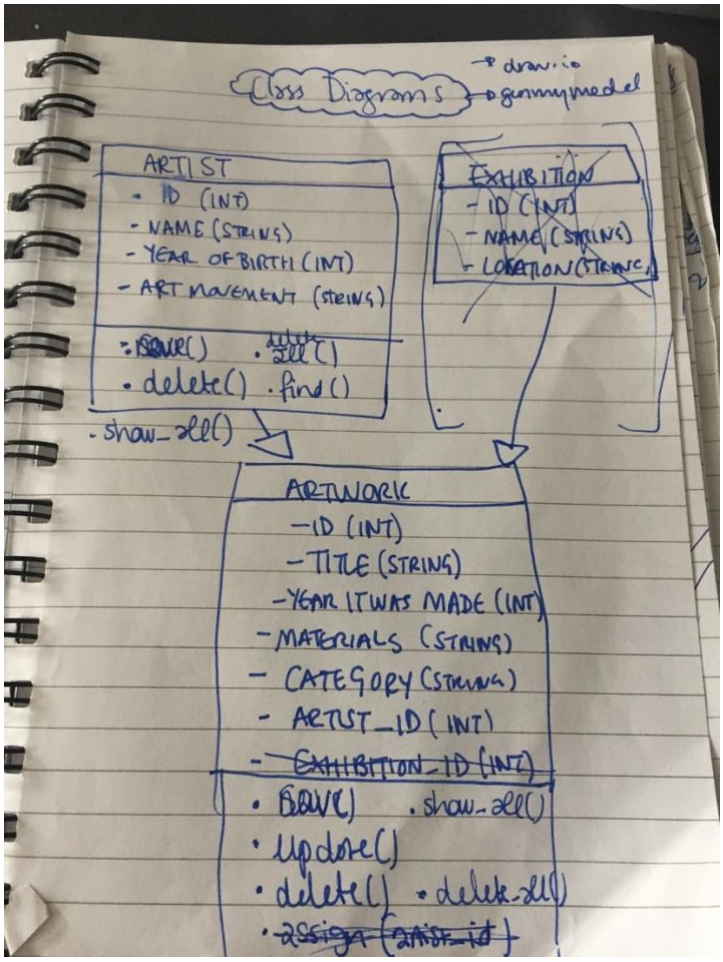
Description here

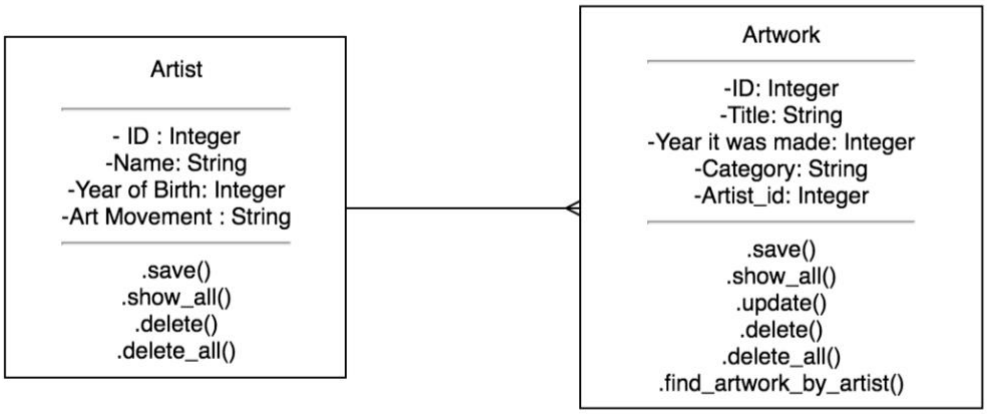
These are screenshots of my solo Ruby project, where I created an art gallery using Ruby, PostgresSQL, HTML & CSS. Below them is a link to view it on Github.

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

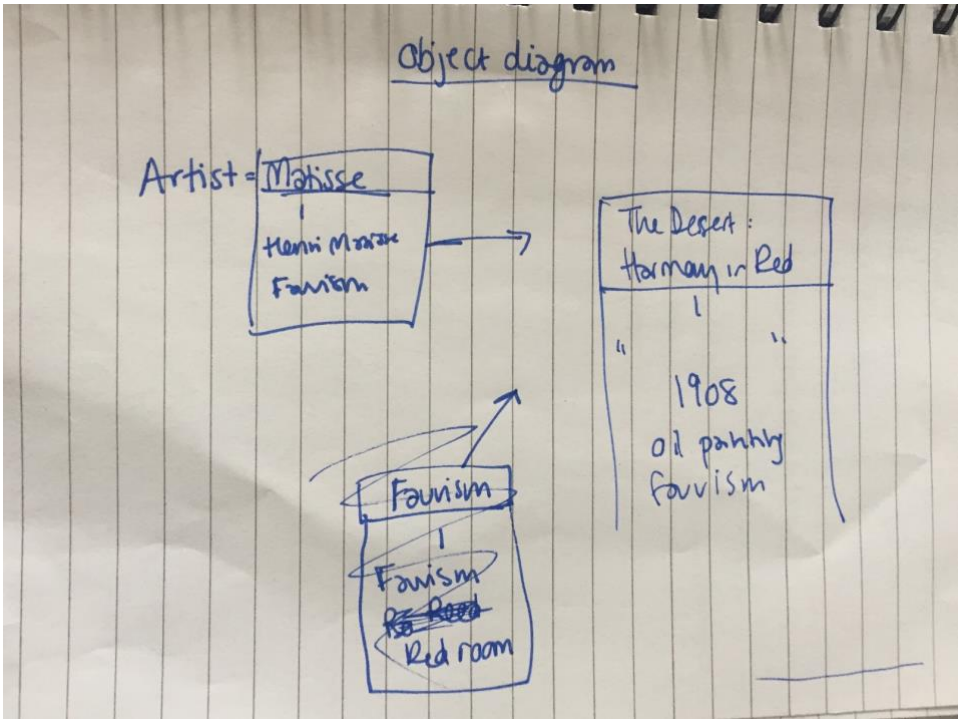
Paste Screenshot here

Class Diagrams





Object Diagrams



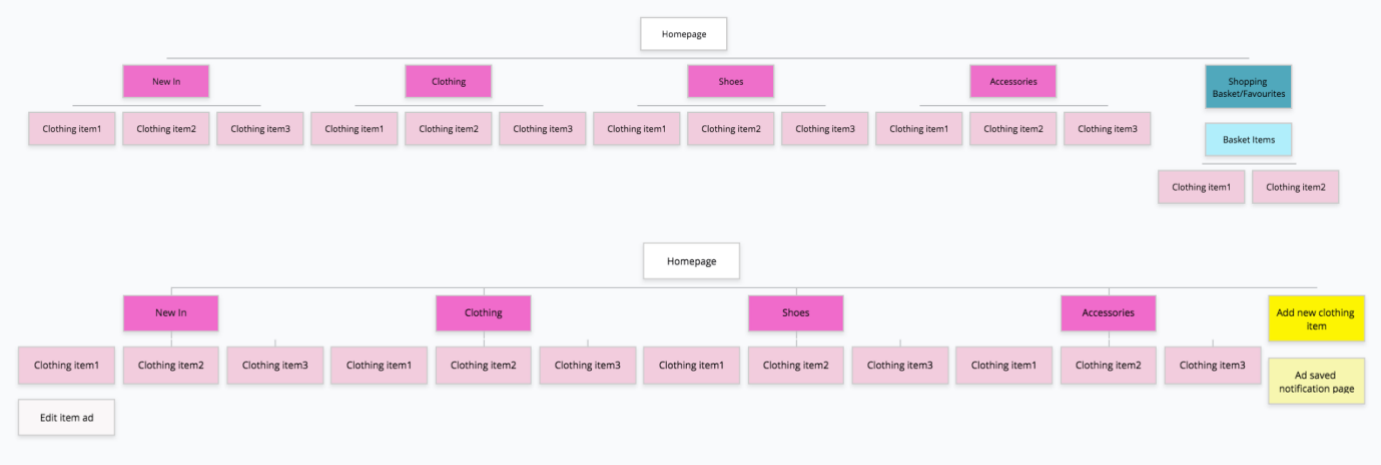
Artist: *Matisse*
- ID : *1* -Name: *Henri Matisse* -Year of Birth: *1869* -Art Movement : *Fauvism*
.save() .show_all() .delete() .delete_all()

Artwork: *The Dessert: Harmony in Red*
-ID: *1* -Title: *The Dessert: Harmony in Red* -Year it was made: *1908* -Category: *Fauvism* -Artist_id: *1*
.save() .show_all() .update() .delete() .delete_all() .find_artwork_by_artist()

Sitemaps

Original:

Split out:



Description here

These are screenshots of my planning initially for my art gallery project and then the final plans before I started coding. I changed my class diagram by cutting my classes down from three to two. I changed my object diagrams by clarifying for myself what an object diagram was. Finally, in my clothing shop project, I changed the sitemap from one map to two as we decided to split out the manager and customer's functionality.

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

Algorithm 1:

```
public void checkGuestIn(Guest guest, Bedroom bedroom) {
    if (guest.countMoney() >= bedroom.getNightlyRate()) {
        guest.money -= bedroom.getNightlyRate();
        bedroom.addGuest(guest);
    }
}
```

This is an algorithm in java to check whether a guest has enough money to rent a room, and if they do, they are added to the room and the nightly rate is taken away from the guest's money.

Algorithm 2:

```
Customer.prototype.sortByMostValuable = function () {
  let sortedArray =  _.sortBy(this.collection, 'price');
  return sortedArray.reverse();
};
```

This algorithm sorts records in a collection by their price using Lodash. Since Lodash default sorts values in ascending order, I then used Javascript's 'reverse' method to get the list in descending order and thus can get the most valuable record by taking the first element in the resulting array.

Week 12

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

Description here

Week 15

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

Description here

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

Paste Screenshot here

Description here

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

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

Paste Screenshot here

Description here

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

Paste Screenshot here

Description here

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

Paste Screenshot here

Description here

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

Paste Screenshot here

Description here