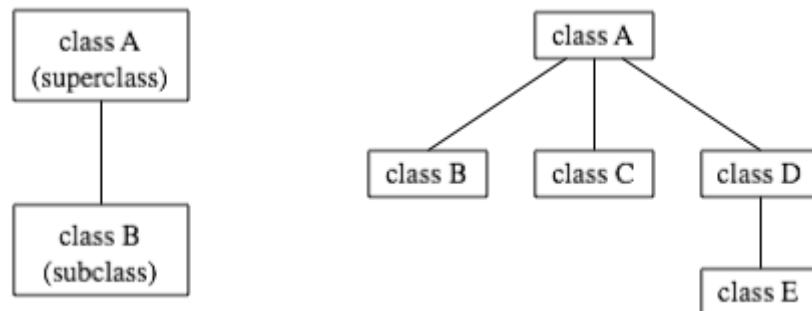


Evidence for project unit I & T

Rob Williams - Cohort E16

Week 8 - Reference A.D 5

An inheritance diagram



Week 8 - Reference I.T 1

An example of encapsulation - From an Instrument class in a music shop, the below image shows that the variables are declared as private then getters are used to be able to access them from outside of the class.

```
public abstract class Instrument implements Playable, Sellable {
    private String manufacturer;
    private int yearOfManufacture;
    private double sellingPrice;
    private double buyingPrice;
    private InstrumentType instrumentType;

    public Instrument(String manufacturer, int yearOfManufacture, double sellingPrice, double buyingPrice, InstrumentType instrumentType) {
        this.manufacturer = manufacturer;
        this.yearOfManufacture = yearOfManufacture;
        this.sellingPrice = sellingPrice;
        this.buyingPrice = buyingPrice;
        this.instrumentType = instrumentType;
    }

    // Getters
    public String getManufacturer() { return manufacturer; }
    public int getYearOfManufacture() { return yearOfManufacture; }
    public double getSellingPrice() { return sellingPrice; }
    public double getBuyingPrice() { return buyingPrice; }
    public InstrumentType getInstrumentType() { return instrumentType; }

    // Methods
    public String play() { return "Playing"; }
    public double calculateMarkup(double sellingPrice, double buyingPrice) {
        return sellingPrice - buyingPrice;
    }
}
```

Week 8 - Reference I.T 2

The use of inheritance in a program - Again from the music shop program is my Instrument superclass.

```
public abstract class Instrument implements Playable, Sellable {  
    private String manufacturer;  
    private int yearOfManufacture;  
    private double sellingPrice;  
    private double buyingPrice;  
    private InstrumentType instrumentType;  
  
    public Instrument(String manufacturer, int yearOfManufacture, double sellingPrice, double buyingPrice, InstrumentType instrumentType) {  
        this.manufacturer = manufacturer;  
        this.yearOfManufacture = yearOfManufacture;  
        this.sellingPrice = sellingPrice;  
        this.buyingPrice = buyingPrice;  
        this.instrumentType = instrumentType;  
    }  
}
```

The following two diagrams show classes for Drum and Guitar, they extend Instrument then use super to access the five generic variables they both need.

```
public class Drums extends Instrument {  
    public int numberInSet;  
    public boolean includesCymbals;  
  
    public Drums(int numberInSet,  
                 boolean includesCymbals,  
                 String manufacturer,  
                 int yearOfManufacture,  
                 double sellingPrice,  
                 double buyingPrice,  
                 InstrumentType instrumentType) {  
        super(manufacturer, yearOfManufacture, sellingPrice, buyingPrice, instrumentType);  
        this.numberInSet = numberInSet;  
        this.includesCymbals = includesCymbals;  
    }  
}
```

```
public class Guitar extends Instrument {  
    public String colour;  
    public String type;  
  
    public Guitar(String colour,  
                  String type,  
                  String manufacturer,  
                  int yearOfManufacture,  
                  double sellingPrice,  
                  double buyingPrice,  
                  InstrumentType instrumentType) {  
        super(manufacturer, yearOfManufacture, sellingPrice, buyingPrice, instrumentType);  
        this.colour = colour;  
        this.type = type;  
    }  
}
```

In the final diagram the tests reference the methods automatically inherited by the individual instruments from the Instrument superclass, all are passing.

```
@Test
public void drumsHaveCymbalsIncluded() {
    assertEquals( expected: true, drums.includesCymbals);
}

@Test
public void guitarHasAManufacturer() {
    assertEquals( expected: "Gibson", guitar.getManufacturer());
}

@Test
public void drumsHaveAYearOfManufacture() {
    assertEquals( expected: 2017, drums.getYearOfManufacture());
}

@Test
public void pianoHasAMarkup() {
    double result = piano.calculateMarkup(piano.getSellingPrice(), piano.getBuyingPrice());
    assertEquals( expected: 700, result, delta: 0.10);
}

@Test
public void drumsHaveAnInstrumentType() {
    assertEquals(InstrumentType.PERCUSSION, drums.getInstrumentType());
}
```

InstrumentTest

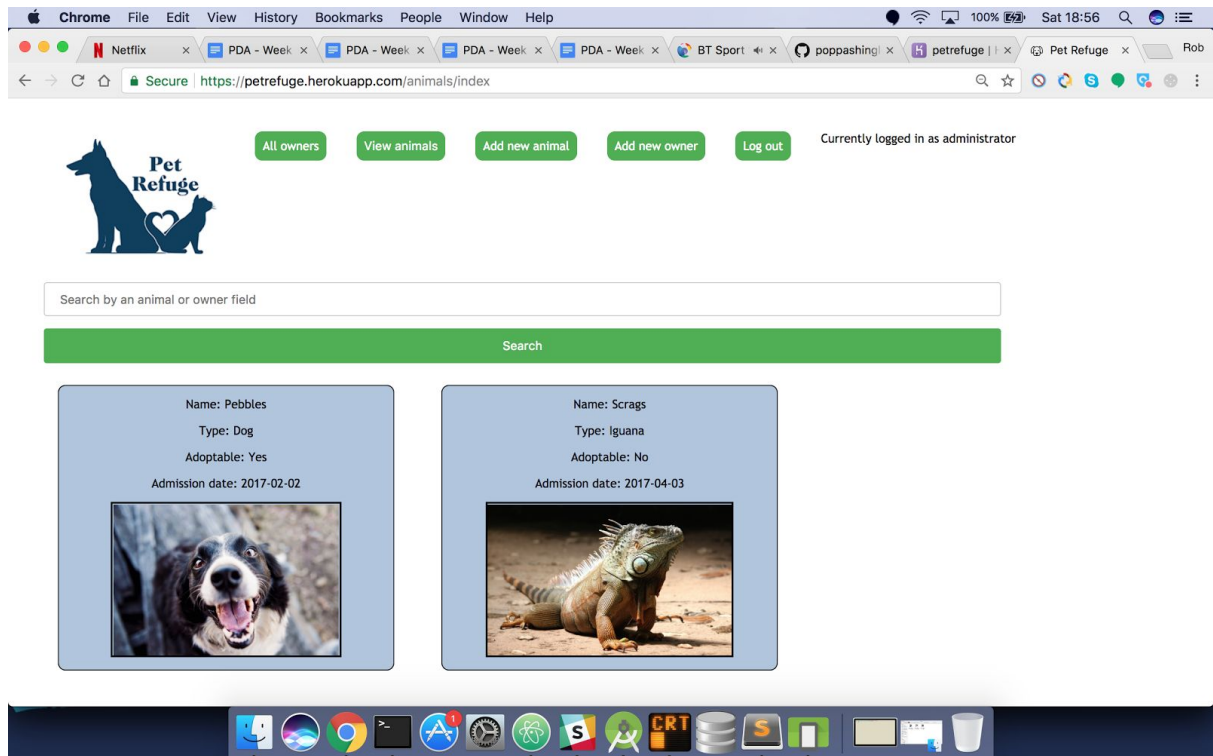
All 8 tests passed – 4ms

Test Name	Duration
InstrumentTest (com.exemplar)	4ms
guitarHasAManufacturer	3ms
pianoHasAMarkup	0ms
drumsHaveCymbalsIncluded	1ms
guitarCanBePlayed	0ms
guitarHasAColour	0ms
pianoHasASize	0ms
drumsHaveAYearOfManufacture	0ms
drumsHaveAnInstrumentType	0ms


Process finished with exit code 0

Week 8 - Reference P 11

Show a screenshot of one of your projects - This is the screenshot of the home page of my Ruby week project, an animal shelter management system. The github link is https://github.com/poppashingles/animal_shelter and link to the live site is <https://dashboard.heroku.com/apps/petrefuge>



Planning screenshots - Below are the screenshots used in my presentation derived from the planning for the animal shelter app. In the end the development exactly followed the planning as I started the coding process with a very clear idea of what I wanted to make and the structure due to the plans I made at the start.

	Caroline Mahoney <ul style="list-style-type: none">• Age: 42• Born and lives in Edinburgh	Behaviours <ul style="list-style-type: none">• Affection for animals led her to volunteer• Occasional use of computers and tech, not massively familiar• Drives her dogs outside the city for long walks
Demographics <ul style="list-style-type: none">• Currently unemployed• Has two dogs• Single• Lives in the city• Well off but wants to keep busy between jobs	Needs and goals <ul style="list-style-type: none">• Something extra on her CV for job seeking through gaining new skills• Would like something that will be easy to use and difficult to make errors on as something small could make a big difference	

User needs

User needs ☆ 🗑 Private

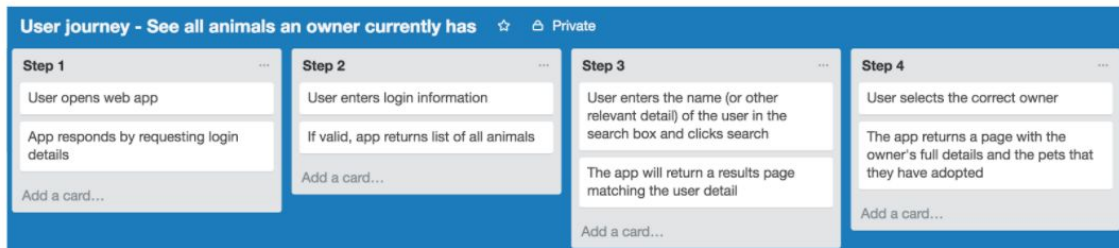
As a...	I want to...	So that...
older man with little tech experience	have an app that is easy to use	I can keep volunteering meaningfully
glasses wearer	see everything clearly	I can be accurate in the system
person who speaks English as a second language	have plain English on the app and visual prompts	I can navigate easily without knowing jargon
person who is not techy	have an app that limits user error	I'm confident using it
Add a card...	Add a card...	Add a card...

Finding all animals of a type

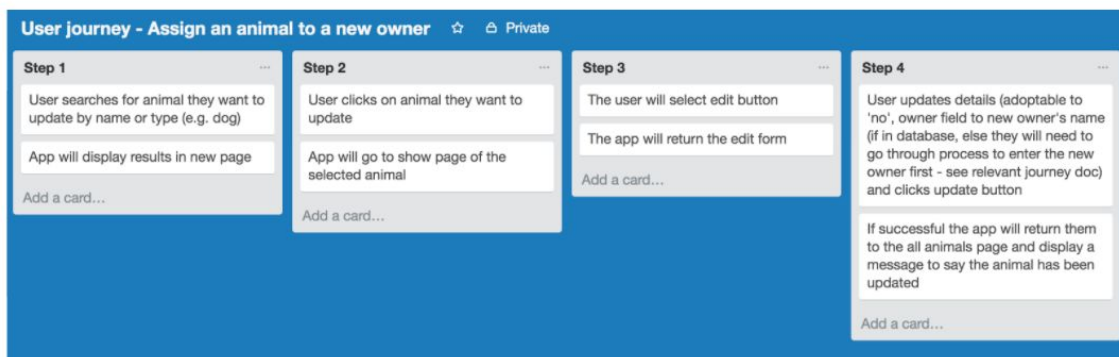
User journey - See all animals of a type ☆ 🗑 Private

Step 1	Step 2	Step 3
User opens web app	User enters login details	User enters the animal type they want to find in the search box and clicks search/hits enter
App responds by requesting login details	If valid, app redirects to list of all animals	App returns a list of all animals of that type in a new page
Add a card...	Add a card...	Add a card...

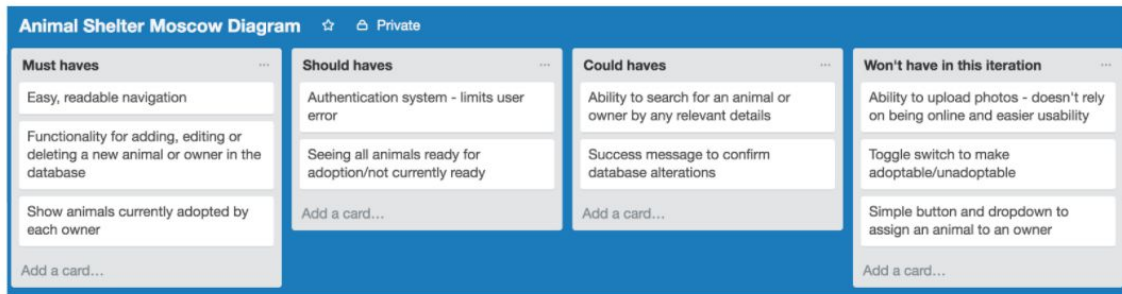
Seeing all animals adopted by an owner



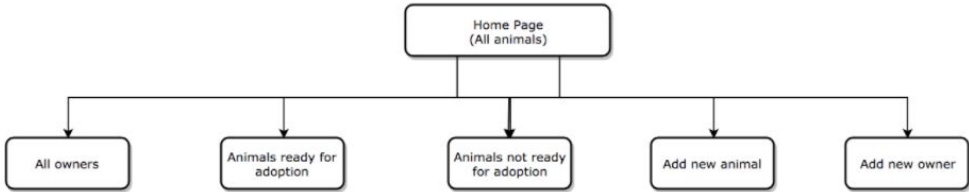
Assigning an animal to a new owner



Moscow diagram



Site Map



Classes

Classes Diagram ☆ Private

Animal properties

id SERIAL4 PRIMARY KEY,

name VARCHAR(255),

type VARCHAR(255),

adoptable BOOLEAN,

admission_date DATE,

photo_url VARCHAR(255),

owner_id INT4 REFERENCES owners(id)

Add a card...

Animal methods

save

update

delete

all

find

find_by_name

find_by_owner

adoptable

owner

Add a card...

Owner properties

id SERIAL4 PRIMARY KEY,

first_name VARCHAR(255),

last_name VARCHAR(255),

address VARCHAR(255),

email VARCHAR(255),

photo_url VARCHAR(255),

phone_number VARCHAR(255)

Add a card...

Owner methods

save

update

delete

all

find

find_by_name

Add a card...