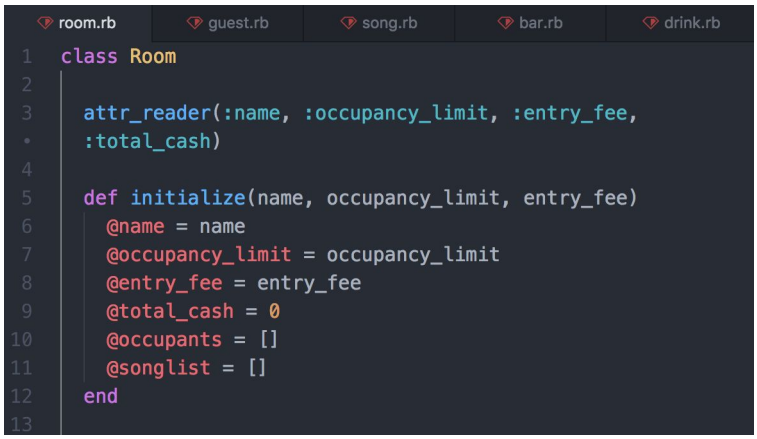
Evidence for Implementation and Testing Unit

Joe Stafford
Cohort: E21

I.T. 5 - Demonstrate the use of an array in a program

- An array in a program (@occupants, @songlist are both arrays in the Room class)



- A function that uses an array (find_guest_by_name method loops through the @occupants array)

- The result of the function running (Below shows successful tests, the arrangement of the array in use and the test I used to test the above method)

```
[➜  weekend_homework_2 git:(master) ruby specs/room_spec.rb
Run options: --seed 54886

# Running:

................

Finished in 0.002159s, 7410.8383 runs/s, 8800.3704 assertions/s.

16 runs, 19 assertions, 0 failures, 0 errors, 0 skips
➜  weekend_homework_2 git:(master)
```

```ruby
class RoomTest < MiniTest::Test

  def setup
    @room = Room.new("Room 1", 3, 10)

    @song = Song.new("Bohemian Rhapsody")

    @guest = Guest.new("Joe Stafford", 20)
    @guest_1 = Guest.new("John Stafford", 20)
    @guest_2 = Guest.new("Joseph Stafford", 20)
    @guest_3 = Guest.new("Jo Stafford", 20)
  end
```

room_spec.rb | guest_spec.rb | song_spec.rb | bar_spec.rb | drink_spec.rb

```ruby
54  end
55
56  def test_can_find_guest_by_name__guest_found
57    @room.add_guest(@guest)
58    assert_equal(@guest, @room.find_guest_by_name("Joe Stafford"))
59  end
60
61  def test_can_find_guest_by_name__guest_not_found
62    @room.add_guest(@guest)
63    assert_nil(nil, @room.find_guest_by_name("Jo Stafford"))
64  end
65
```

I.T. 6 - Demonstrate the use of a hash in a program

- A hash in a program (a hash of a pet, with several key - value pairs)

```ruby
@new_pet = {
        name: "Bors the Younger",
        pet_type: :cat,
        breed: "Cornish Rex",
        price: 100
      }
```

- A function that uses a hash (this method adds the new_pet hash to the customers array
-

```ruby
def add_pet_to_customer(customer, new_pet)
  customer[:pets].push(new_pet)
end
```

- The result of the function running (Below shows successful tests and the test I used to test the above method)

```
[➜ weekend_homework_1 git:(master) ruby spec
s/pet_shop_spec.rb
Run options: --seed 45031

# Running:

......................

Finished in 0.002424s, 9075.9074 runs/s, 127
88.7785 assertions/s.

22 runs, 31 assertions, 0 failures, 0 errors
, 0 skips
```

```ruby
def test_add_pet_to_customer
  customer = @customers[0]
  add_pet_to_customer(customer, @new_pet)
  assert_equal(1, customer_pet_count(customer))
end
```

I.T. 3 - Demonstrate searching data in a program

- Function that searches data (The below method utilises an SqlRunner to run the SQL statement ‘SELECT’)

```ruby
62
63  def self.all()
64    sql = "SELECT * FROM customers"
65    customer_data = SqlRunner.run(sql)
66    return
      Customer.map_items(customer_data)
67  end
68
```

- The result of the functioning program (Below shows the array of hashes returned when calling the function above)

```
[1] pry(main)> Customer.all()
 => [#<Customer:0x00007f8d94a24170
   @funds=10,
   @id=7,
   @name="Joe Stafford">,
  #<Customer:0x00007f8d94a240a8
   @funds=5,
   @id=8,
   @name="Euan Bell">,
  #<Customer:0x00007f8d931abf80
   @funds=20,
   @id=9,
   @name="Ruairidh Grass">,
  #<Customer:0x00007f8d931abe68
   @funds=10,
   @id=10,
   @name="Paul Stevenson">]
 [2] pry(main)>
```

I.T. 4 - Demonstrate sorting data in a program

- Function that sorts data (below function shows sorting data using the id of a customer to select all films that customer is going to see)

```ruby
def films()
  sql = "SELECT films.*
  FROM films
  INNER JOIN tickets
  ON tickets.film_id = films.id
  WHERE customer_id = $1"
  values = [@id]
  film_data = SqlRunner.run(sql, values)
  return Film.map_items(film_data)
end
```

- The result of the function running (this shows an array of hashes showing the films that the object ‘customer1’ is going to see)

```
[2] pry(main)> customer1.films()
=> [#<Film:0x00007f8d9412f498
  @id=7,
  @price=2,
  @title="Dr Strangelove">,
 #<Film:0x00007f8d9412f3d0
  @id=8,
  @price=1,
  @title="The Ladykillers">]
```

I.T. 7 - Demonstrate the use of Polymorphism in a program

- Above shows an abstract super class ‘Animal Attraction’, and two sub-classes, ‘Avian’ and ‘Terrestial’, which extend the super class.

```
public abstract class AnimalAttraction implements IEdible {
    private String genera;
    private DietType diet;
    private int caloricContent;
    private int requiredDailyCalories;
    private boolean enraged;
    private boolean plantBased;
    private ArrayList<IEdible> stomach;

    public AnimalAttraction(String genera, DietType diet, int caloricContent, int requiredDailyCalories) {
        this.genera = genera;
        this.diet = diet;
        this.caloricContent = caloricContent;
        this.requiredDailyCalories = requiredDailyCalories;
        this.stomach = new ArrayList<>();
        this.plantBased = false;
    }


public class Avian extends AnimalAttraction {

    public Avian(String genera, DietType diet, int caloricContent, int requiredDailyCalories) {
        super(genera, diet, caloricContent, requiredDailyCalories);
    }
}
```

- An object of type Avian is both an ‘Avian’ and an ‘Animal Attraction’.

I.T. 1 Encapsulation in a program

```
public abstract class AnimalAttraction implements IEdible {
    private String genera;
    private DietType diet;
    private int caloricContent;
    private int requiredDailyCalories;
    private boolean enraged;
    private boolean plantBased;
    private ArrayList<IEdible> stomach;

    public AnimalAttraction(String genera, DietType diet, int caloricContent, int requiredDailyCalories) {
        this.genera = genera;
        this.diet = diet;
        this.caloricContent = caloricContent;
        this.requiredDailyCalories = requiredDailyCalories;
        this.stomach = new ArrayList<>();
        this.plantBased = false;
    }

    public String getGenera() {
        return this.genera;
    }

    public DietType getDiet() {
        return this.diet;
    }

    public int getCaloricContent() {
        return this.caloricContent;
    }

    public void setCaloricContent(int newCaloricContent) {
        this.caloricContent = newCaloricContent;
    }

    public int getRequiredDailyCalories() {
        return this.requiredDailyCalories;
    }
```

- The above code shows several private variables, and a sample of getter and setter methods used to get/set the private variables

I.T. 2 - Use of inheritance in a program

- Super class

```
public abstract class AnimalAttraction implements IEdible {
    private String genera;
    private DietType diet;
    private int caloricContent;
    private int requiredDailyCalories;
    private boolean enraged;
    private boolean plantBased;
    private ArrayList<IEdible> stomach;

    public AnimalAttraction(String genera, DietType diet, int caloricContent, int requiredDailyCalories) {
        this.genera = genera;
        this.diet = diet;
        this.caloricContent = caloricContent;
        this.requiredDailyCalories = requiredDailyCalories;
        this.stomach = new ArrayList<>();
        this.plantBased = false;
    }
```

- Class that inherits from above super class

```
public class Avian extends AnimalAttraction {

    public Avian(String genera, DietType diet, int caloricContent, int requiredDailyCalories) {
        super(genera, diet, caloricContent, requiredDailyCalories);
    }
}
```

- An object of above class

```
private Terrestrial herbivoreTerrestrial1;
```

- A method that uses the information inherited from another class

```
@Test
public void canSetCaloricContent() {
    omnivoreTerrestrial.setCaloricContent(900);
    assertEquals( expected: 900, omnivoreTerrestrial.getCaloricContent());
}
```