

Evidence for Implementation & Testing Unit

Mark Blanford
Cohort E18

I.T.1 Example of Encapsulation

```
8
9     public class Guitar implements Serializable{
10
11         private String brand;
12         private String model;
13         private String colour;
14         private int image_id;
15
16         public Guitar(String brand, String model, String colour, int image_id){
17             this.brand = brand;
18             this.model = model;
19             this.colour = colour;
20             this.image_id = image_id;
21         }
22
23
24         public String getBrand() {
25             return this.brand;
26         }
27
28         public String getModel() {
29             return this.model;
30         }
31
32         public String getColour() {
33             return this.colour;
34         }
35
36         public int getImage_id(){
37             return this.image_id;
38         }
39     }
```

I.T.2 Example of Inheritance

An abstract class, SellableItem:

```
1  package Shop;
2
3  import Interfaces.ISellable;
4
5  public abstract class SellableItem implements ISellable{
6
7      private double buyPrice;
8      private double sellPrice;
9
10     public SellableItem(double buyPrice, double sellPrice){
11         this.buyPrice = buyPrice;
12         this.sellPrice = sellPrice;
13     }
14
15     public double getBuyPrice() {
16         return this.buyPrice;
17     }
18
19     public double getSellPrice() {
20         return this.sellPrice;
21     }
22
23     public double calculateMarkup(){
24         double markUp = getSellPrice() - getBuyPrice();
25         return markUp;
26     }
27
28 }
```

A class that inherits from the SellableItem class:

```
1  package Instruments;
2
3  import Shop.SellableItem;
4
5  public abstract class Instrument extends SellableItem{
6
7      private InstrumentType instrumentType;
8      private String brand;
9
10     public Instrument(InstrumentType instrumentType, String brand, double buyPrice, double sellPrice){
11         super(buyPrice, sellPrice);
12         this.instrumentType = instrumentType;
13         this.brand = brand;
14     }
15
16     public InstrumentType getInstrumentType() { return this.instrumentType; }
17
18     public String getBrand(){
19         return this.brand;
20     }
21
22 }
23
24
25 }
```

A class inheriting from the Instrument (and SellableItem) class:

```
1 package Instruments;
2
3 import ...
4
5 public class Guitar extends Instrument implements IColour, IPlayable{
6
7     private String colour;
8     private GuitarType guitarType;
9
10    public Guitar(InstrumentType instrumentType, GuitarType guitarType, String brand, String colour, double buyPrice, do
11        super(instrumentType, brand, buyPrice, sellPrice);
12        this.colour = colour;
13        this.guitarType = guitarType;
14    }
15
16    public String getColour(){
17        return this.colour;
18    }
19
20    public GuitarType getGuitarType(){
21        return this.guitarType;
22    }
23
24    public String play(){
25        return "Brrringgg!";
26    }
27
28    public int getNumberOfStrings() {
29        return guitarType.getNumberOfStrings();
30    }
31
32
33
34 }
```

An object of the Guitar class demonstrating inherited methods:

```
1 package InstrumentTests;
2
3 import ...
4
5 public class GuitarTest {
6
7     Guitar guitar;
8
9     @Before
10    public void before(){
11        guitar = new Guitar(InstrumentType.STRINGED, GuitarType.ELECTRIC6,
12            | brand: "Hagstrom", colour: "Satin Black", buyPrice: 300.00, sellPrice: 380.99);
13    }
14
15    @Test
16    public void hasType(){
17        assertEquals(InstrumentType.STRINGED, guitar.getInstrumentType());
18    }
19
20    @Test
21    public void hasBrand(){
22        assertEquals("Hagstrom", guitar.getBrand());
23    }
24
25    @Test
26    public void hasColour(){
27        assertEquals("Satin Black", guitar.getColour());
28    }
29
30    @Test
31    public void hasBuyPrice(){
32        assertEquals(300.00, guitar.getBuyPrice(), delta: 0.01);
33    }
34
35    @Test
36    public void hasSellPrice(){
37        assertEquals(380.99, guitar.getSellPrice(), delta: 0.01);
38    }
39
40
41    @Test
42    public void hasToString(){
43        assertEquals("Guitar{brand='Hagstrom', colour='Satin Black', buyPrice=300.0, sellPrice=380.99, type=Electric 6, strings=6}", guitar.toString());
44    }
45}
```

I.T.3 Example of Searching

Creating and saving Artist and Album objects to a database:

```
 8 ~ artist1 = Artist.new({
 9     'name' => 'Biffy Clyro'
10   })
11
12 ~ artist2 = Artist.new({
13     'name' => 'Arcane Roots'
14   })
15
16   artist1.save()
17   artist2.save()
18
19 # Artist.delete('Biffy Clyro', artist1.id)
20
21 ~ album1 = Album.new({
22     'title' => 'The Vertigo of Bliss',
23     'genre' => 'Alternative Rock',
24     'artist_id' => artist1.id
25   })
26
27 ~ album2 = Album.new({
28     'title' => 'Opposites',
29     'genre' => 'Alternative Rock',
30     'artist_id' => artist1.id
31   })
32
33 ~ album3 = Album.new({
34     'title' => 'Melancholia Hymns',
35     'genre' => 'Progressive Post-Hardcore',
36     'artist_id' => artist2.id
37   })
38
39   album1.save()
40   album2.save()
41   album3.save()
```

Artist find_by_id function to find the artist in the database:

```
68   def Artist.find_by_id(id)
69     sql = "SELECT * FROM artists WHERE id = $1;"
70     values = [id]
71     found_artist = SqlRunner.run(sql, values)
72     return found_artist[0]
73   end
74
```

Album find_by_id function to find the album in the database:

```
59
60     def Album.find_by_id(id)
61         sql = "SELECT * FROM albums WHERE id = $1"
62         values = [id]
63         found_album = SqlRunner.run(sql, values)
64         return found_album[0]
65     end
66
```

Function calls:

```
61
62     artist_albums = artist1.albums()
63     artist_albums2 = artist2.albums()
64
65     album_artist = album1.artist()
66
67     find_artist = Artist.find_by_id(artist1.id)
68     find_album = Album.find_by_id(album2.id)
69
```

Function results:

```
[→ music_library git:(master) ✘ ruby console.rb      ]
From: /Users/user/codeclan_work/e18_homework/week_03/
day_03/music_library/console.rb @ line 71 :

66:
67:   find_artist = Artist.find_by_id(artist1.id)
68:   find_album = Album.find_by_id(album2.id)
69:
70:   binding.pry
=> 71: nil

[[1] pry(main)> find_artist
=> {"id"=>"136", "name"=>"Biffy Clyro"}
[[2] pry(main)> find_album
=> {"id"=>"178",
    "title"=>"Opposites",
    "genre"=>"Alternative Rock",
    "artist_id"=>"136"}
[3] pry(main)> ]
```

I.T.4 Example of Sorting

A function that selects albums and artist names from a database and sorts them alphabetically according to the artist name:

```
66  def self.all()
67    sql = 'SELECT albums.*, artists.name FROM albums
68      INNER JOIN artists
69        ON artists.id = albums.artist_id
70        ORDER BY artists.name;'
71    all_albums_array = SqlRunner.run(sql)
72    all_albums = all_albums_array.map { |album| Album.new(album) }
73    return all_albums
74  end
75
```

The route that uses this function:

```
6  get('/albums') do
7    @albums = Album.all()
8    erb(:"albums/index")
9  end
10
```

The displayed result of this sorted database information:

Artist	Album
Arcane Roots	Blood and Chemistry
Arcane Roots	Melancholia Hymns
Biffy Clyro	Only Revolutions
Big Thief	Masterpiece
Black Peaks	Statues
Rachel Sermanni	Under Mountains
Rachel Sermanni	Tied to the Moon
Terrestria	Things You'll Never Know

I.T.5 Example of Array Use

A class (Room) with a property that is an array of Song objects (playlist):

```
1  class Room
2
3  attr_reader :playlist, :capacity, :guests, :entry_fee
4  attr_accessor :till
5
6  def initialize(playlist, capacity)
7      @playlist = playlist
8      @capacity = capacity
9      @guests = []
10     @entry_fee = 15
11     @till = 0
12   end
13
14  def add_guest(guest)
15      if @capacity > 0 && guest.wallet() >= @entry_fee
16          @guests << guest
17          @capacity -= 1
18          @till += 15
19          take_money(guest, 15)
20          guest.favourite_song_on_playlist(@playlist, guest.favourite)
21      else
22          return "I'm sorry, you can't come in, #{guest.name()}."
23      end
24  end
25
26  def take_money(guest, amount)
27      guest.wallet -= amount
28  end
29
30  def add_to_playlist(song)
31      @playlist << song
32  end
33
34 end
```

An object of the Room class with a playlist array passed in at instantiation:

```
room_spec.rb
1 require("minitest/autorun")
2 require("minitest/rg")
3 require_relative("../room.rb")
4 require_relative("../song.rb")
5 require_relative("../guest.rb")
6
7 class TestRoom < Minitest::Test
8
9     def setup
10         @song1 = Song.new("Ignition: Remix")
11         @song2 = Song.new("Lose Yourself")
12         @song3 = Song.new("Screamager")
13         @song4 = Song.new("Blank Space")
14         @song5 = Song.new("Glitter and Trauma")
15         @playlist = [@song1, @song2, @song3, @song4, @song5]
16         @song6 = Song.new("The Fog")
17         @room1 = Room.new(@playlist, 5)
```

A test using a function that appends an item to the array:

```
72
73
43
44 def test_add_to_playlist
45     @room1.add_to_playlist(@song6)
46     expected = [@song1, @song2, @song3, @song4, @song5, @song6]
47     actual = @room1.playlist
48     assert_equal(expected, actual)
49 end
50
```

The result of this test (plus 10 others) running:

```
[→ weekend_homework cd karaoke_bar ]
[→ karaoke_bar git:(master) ruby specs/room_spec.rb ]
Run options: --seed 3988

# Running:

.....
Finished in 0.001266s, 6319.1160 runs/s, 8688.7845 assertions/s.

8 runs, 11 assertions, 0 failures, 0 errors, 0 skips
→ karaoke_bar git:(master) ]
```

I.T.6 Example of Hash Use

Customer class that takes in an options hash on instantiation:

```
(customers.rb)
1 require_relative('../db/sql_runner.rb')
2 require_relative('films.rb')
3 require('pry-byebug')
4
5 class Customer
6
7   attr_accessor :name, :funds
8   attr_reader :id
9
10  def initialize(options)
11    @id = options['id'].to_i
12    @name = options['name']
13    @funds = options['funds'].to_i
14  end
15
16  def save()
17    sql = "INSERT INTO customers (name, funds)
18      VALUES ($1, $2)
19      RETURNING id;"
20    values = [@name, @funds]
21    customer_hash_array = SqlRunner.run(sql, values)
22    @id = customer_hash_array[0]['id'].to_i
23  end
24
25  def update()
26    sql = "UPDATE customers SET (name, funds)
27      = ($1, $2)
28      WHERE id = $3;"
29    values = [@name, @funds, @id]
30    SqlRunner.run(sql, values)
31  end
32
33  def delete()
34    sql = "DELETE FROM customers
```

Instantiation of Customer objects (and Film objects) with hashes:

```
1 require('pry-bybug')
2 require_relative('models/customers.rb')
3 require_relative('models/films.rb')
4 require_relative('models/tickets.rb')
5
6 Ticket.delete_all()
7 Film.delete_all()
8 Customer.delete_all()
9
10 customer1 = Customer.new({'name' => 'Mark Blanford', 'funds' => 35})
11 customer1.save()
12 customer2 = Customer.new({'name' => 'Aline Nardo', 'funds' => 83})
13 customer2.save()
14 customer3 = Customer.new({'name' => 'Ciaran McKenna', 'funds' => 50})
15 customer3.save()
16
17 film1 = Film.new({'title' => 'Let The Right One In', 'price' => 8})
18 film1.save()
19 film2 = Film.new({'title' => 'Ghostbusters', 'price' => 5})
20 film2.save()
21 film3 = Film.new({'title' => 'Predator', "price" => 4})
22 film3.save()
23
```

Function to return all Customer objects from a database (as hash-like objects) and instantiate them as new objects:

```
64
65   def self.all()
66     sql = "SELECT * FROM customers;"
67     all_customers_hashes = SqlRunner.run(sql)
68     all_customers = all_customers_hashes.map() { |customer_hash| Customer.new(customer_hash)}
69     return all_customers
70   end
71
```

Calling the above function:

```
41   customer_list = Customer.all()
```

The result of the Customer.all function:

```
[→ cinema git:(master) ✘ ruby console.rb ]  
  
From: /Users/user/codeclan_work/e18_homework/week_03/  
weekend_homework/cinema/console.rb @ line 66 :  
  
  61: film3_customers = film3.customers()  
  62: number_tickets_bought = customer1.number_of_t  
ickets()  
  63: number_customers_film1 = film1.number_of_cust  
omers()  
  64:  
  65: binding.pry  
=> 66: nil  
  
[[1] pry(main)> ls ]  
self.methods: inspect to_s  
locals:  
- customer2 film_list  
-- customer3 number_customers_film1  
_dir_ customer_films number_tickets_bought  
_ex_ customer_list ticket_list  
_file_ film1 updated_customer_list  
_in_ film1_customers updated_film_list  
_out_ film2 updated_ticket_list  
_pry_ film3  
customer1 film3_customers  
[[2] pry(main)> customer_list ]  
=> [#<Customer:0x007fc18f54e180  
@funds=83,  
@id=81,  
@name="Aline Nardo",  
#<Customer:0x007fc18f54e0b8  
@funds=50,  
@id=82,  
@name="Ciaran McKenna",  
#<Customer:0x007fc18f54dff0  
@funds=55,  
@id=80,  
@name="Mark E Blanford">]  
[3] pry(main)> █
```

I.T.7 Example of Polymorphism

An interface, IConnect, in Java:

The screenshot shows a Java code editor with three tabs: Desktop.java, ComputerTest.java, and Network.java. The Desktop.java tab is active, displaying the following code:

```
1 public interface IConnect {  
2     String connect(String data);  
3 }  
4  
5  
6
```

Classes implementing the IConnect interface:

The screenshot shows a Java code editor with four tabs: Desktop.java, ComputerTest.java, Network.java, and NetworkTest.java. The Desktop.java tab is active, displaying the following code:

```
1 public class Desktop implements IConnect {  
2     private String name;  
3     private String make;  
4     private String model;  
5  
6     public Desktop(String name, String make, String model) {  
7         this.name = name;  
8         this.make = make;  
9         this.model = model;  
10    }  
11  
12    public String getName() {  
13        return name;  
14    }  
15  
16    public String getMake() {  
17        return make;  
18    }  
19  
20    public String getModel() {  
21        return model;  
22    }  
23  
24    public String connect(String networkName){  
25        return "Desktop connecting to network: " + networkName;  
26    }  
27  
28 }
```

The screenshot shows a Java code editor with multiple tabs at the top: Desktop.java, ComputerTest.java, Network.java, NetworkTest.java, and InternetRadio.java. The InternetRadio.java tab is active. The code implements the IConnect interface:

```
1 public class InternetRadio implements IConnect {  
2     public String connect(String networkName){  
3         return "Internet radio connecting to network: " + networkName;  
4     }  
5     public String tune(String station){  
6         return "Tuning to: " + station;  
7     }  
8 }  
9  
10  
11
```

The screenshot shows a Java code editor with multiple tabs at the top: Desktop.java, ComputerTest.java, Network.java, Printer.java, NetworkTest.java, InternetRadio.java, and IConnect.java. The Printer.java tab is active. The code implements the IConnect interface:

```
1 public class Printer implements IConnect {  
2     public String print(String data) { return "printing: " + data; }  
3     public String connect(String networkName) { return "Printer connecting to network: " + networkName; }  
4 }  
5  
6  
7  
8  
9  
10  
11  
12
```

A class (Network) that collects objects of the above polymorphic (IConnect) classes in an ArrayList:

```
1 import java.util.*;
2
3 public class Network {
4     private String name;
5     private ArrayList<IConnect> devices;
6     private int slots;
7
8     public Network(String name, int slots){
9         this.devices = new ArrayList<IConnect>();
10        this.name = name;
11        this.slots = slots;
12    }
13
14    public String getName(){
15        return name;
16    }
17
18    public int getCapacity(){
19        return this.slots;
20    }
21
22    public int deviceCount(){
23        return devices.size();
24    }
25
26    public void connect(IConnect device){
27        if((slots - this.devices.size()) > 0) {
28            devices.add(device);
29        }
30    }
31
32    public void disconnectAll(){
33        devices.clear();
34    }
35
36    public int listFreeSlots() {
37        return slots - this.devices.size();
38    }
39}
```