Evidence for Implementation & Testing Unit

Mark Blanford
Cohort E18

I.T.1 Example of Encapsulation

```
public class Guitar implements Serializable{

    private String brand;
    private String model;
    private String colour;
    private int image_id;

    public Guitar(String brand, String model, String colour, int image_id){
        this.brand = brand;
        this.model = model;
        this.colour = colour;
        this.image_id = image_id;
    }


    public String getBrand() {
        return this.brand;
    }

    public String getModel() {
        return this.model;
    }

    public String getColour() {
        return this.colour;
    }

    public int getImage_id(){
        return this.image_id;
    }
}
```

I.T.2 Example of Inheritance

An abstract class, SellableItem:

```java
package Shop;

import Interfaces.ISellable;

public abstract class SellableItem implements ISellable{

    private double buyPrice;
    private double sellPrice;

    public SellableItem(double buyPrice, double sellPrice){
        this.buyPrice = buyPrice;
        this.sellPrice = sellPrice;
    }

    public double getBuyPrice() {
        return this.buyPrice;
    }

    public double getSellPrice() {
        return this.sellPrice;
    }

    public double calculateMarkup(){
        double markUp = getSellPrice() - getBuyPrice();
        return markUp;
    }
}
```

A class that inherits from the SellableItem class:

```java
package Instruments;

import Shop.SellableItem;

public abstract class Instrument extends SellableItem{

    private InstrumentType instrumentType;
    private String brand;

    public Instrument(InstrumentType instrumentType, String brand, double buyPrice, double sellPrice){
        super(buyPrice, sellPrice);
        this.instrumentType = instrumentType;
        this.brand = brand;
    }

    public InstrumentType getInstrumentType() { return this.instrumentType; }

    public String getBrand(){
        return this.brand;
    }

}
```

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

```java
package Instruments;

import ...

public class Guitar extends Instrument implements IColour, IPlayable{

    private String colour;
    private GuitarType guitarType;

    public Guitar(InstrumentType instrumentType, GuitarType guitarType, String brand, String colour, double buyPrice, do
        super(instrumentType, brand, buyPrice, sellPrice);
        this.colour = colour;
        this.guitarType = guitarType;
    }

    public String getColour(){
        return this.colour;
    }

    public GuitarType getGuitarType(){
        return this.guitarType;
    }

    public String play(){
        return "Brrringgg!";
    }

    public int getNumberOfStrings() {
        return guitarType.getNumberOfStrings();
    }

}
```

An object of the Guitar class demonstrating inherited methods:

```java
package InstrumentTests;

import ...

public class GuitarTest {

    Guitar guitar;

    @Before
    public void before(){
        guitar = new Guitar(InstrumentType.STRINGED, GuitarType.ELECTRIC6,
                brand: "Hagstrom",  colour: "Satin Black",  buyPrice: 300.00,  sellPrice: 380.99);
    }

    @Test
    public void hasType(){
        assertEquals(InstrumentType.STRINGED, guitar.getInstrumentType());
    }

    @Test
    public void hasBrand(){
        assertEquals( expected: "Hagstrom", guitar.getBrand());
    }

    @Test
    public void hasColour(){
        assertEquals( expected: "Satin Black", guitar.getColour());
    }

    @Test
    public void hasBuyPrice(){
        assertEquals( expected: 300.00, guitar.getBuyPrice(),  delta: 0.01);
    }

    @Test
    public void hasSellPrice(){
        assertEquals( expected: 380.99, guitar.getSellPrice(),  delta: 0.01);
    }
```

I.T.3 Example of Searching

Creating and saving Artist and Album objects to a database:

```
artist1 = Artist.new({
  'name' => 'Biffy Clyro'
})

artist2 = Artist.new({
  'name' => 'Arcane Roots'
})

artist1.save()
artist2.save()

# Artist.delete('Biffy Clyro', artist1.id)

album1 = Album.new({
  'title' => 'The Vertigo of Bliss',
  'genre' => 'Alternative Rock',
  'artist_id' => artist1.id
  })

  album2 = Album.new({
    'title' => 'Opposites',
    'genre' => 'Alternative Rock',
    'artist_id' => artist1.id
    })

  album3 = Album.new({
    'title' => 'Melancholia Hymns',
    'genre' => 'Progressive Post-Hardcore',
    'artist_id' => artist2.id
    })

album1.save()
album2.save()
album3.save()
```

Artist find_by_id function to find the artist in the database:

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

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
```

The route that uses this function:

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

The displayed result of this sorted database information:

I.T.5 Example of Array Use

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

```ruby
class Room

attr_reader :playlist, :capacity, :guests, :entry_fee
attr_accessor :till

  def initialize(playlist, capacity)
    @playlist = playlist
    @capacity = capacity
    @guests = []
    @entry_fee = 15
    @till = 0
  end

  def add_guest(guest)
    if @capacity > 0 && guest.wallet() >= @entry_fee
      @guests << guest
      @capacity -= 1
      @till += 15
      take_money(guest, 15)
      guest.favourite_song_on_playlist(@playlist, guest.favourit
    else
      return "I'm sorry, you can't come in, #{guest.name()}."
    end
  end

  def take_money(guest, amount)
    guest.wallet -= amount
  end

  def add_to_playlist(song)
    @playlist << song
  end

end
```

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

room_spec.rb

```ruby
require("minitest/autorun")
require("minitest/rg")
require_relative("../room.rb")
require_relative("../song.rb")
require_relative("../guest.rb")

class TestRoom < Minitest::Test

  def setup
    @song1 = Song.new("Ignition: Remix")
    @song2 = Song.new("Lose Yourself")
    @song3 = Song.new("Screamager")
    @song4 = Song.new("Blank Space")
    @song5 = Song.new("Glitter and Trauma")
    @playlist = [@song1, @song2, @song3, @song4, @song5]
    @song6 = Song.new("The Fog")
    @room1 = Room.new(@playlist, 5)
```

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

```ruby
  def test_add_to_playlist
    @room1.add_to_playlist(@song6)
    expected = [@song1, @song2, @song3, @song4, @song5, @song6]
    actual = @room1.playlist
    assert_equal(expected, actual)
  end
```

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 as
sertions/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

```ruby
require_relative('../db/sql_runner.rb')
require_relative('films.rb')
require('pry-byebug')

class Customer

  attr_accessor :name, :funds
  attr_reader :id

  def initialize(options)
    @id = options['id'].to_i
    @name = options['name']
    @funds = options['funds'].to_i
  end

  def save()
    sql = "INSERT INTO customers (name, funds)
    VALUES ($1, $2)
    RETURNING id;"
    values = [@name, @funds]
    customer_hash_array = SqlRunner.run(sql, values)
    @id = customer_hash_array[0]['id'].to_i
  end

  def update()
    sql = "UPDATE customers SET (name, funds)
    = ($1, $2)
    WHERE id = $3;"
    values = [@name, @funds, @id]
    SqlRunner.run(sql, values)
  end

  def delete()
    sql = "DELETE FROM customers
```

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

```
require('pry-byebug')
require_relative('models/customers.rb')
require_relative('models/films.rb')
require_relative('models/tickets.rb')

Ticket.delete_all()
Film.delete_all()
Customer.delete_all()

customer1 = Customer.new({'name' => 'Mark Blanford', 'funds' => 35})
customer1.save()
customer2 = Customer.new({'name' => 'Aline Nardo', 'funds' => 83})
customer2.save()
customer3 = Customer.new({'name' => 'Ciaran McKenna', 'funds' => 50})
customer3.save()

film1 = Film.new({'title' => 'Let The Right One In', 'price' => 8})
film1.save()
film2 = Film.new({'title' => 'Ghostbusters', 'price' => 5})
film2.save()
film3 = Film.new({'title' => 'Predator', "price" => 4})
film3.save()
```

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

```
def self.all()
  sql = "SELECT * FROM customers;"
  all_customers_hashes = SqlRunner.run(sql)
  all_customers = all_customers_hashes.map() { |customer_hash| Customer.new(customer_hash)}
  return all_customers
end
```

Calling the above function:

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

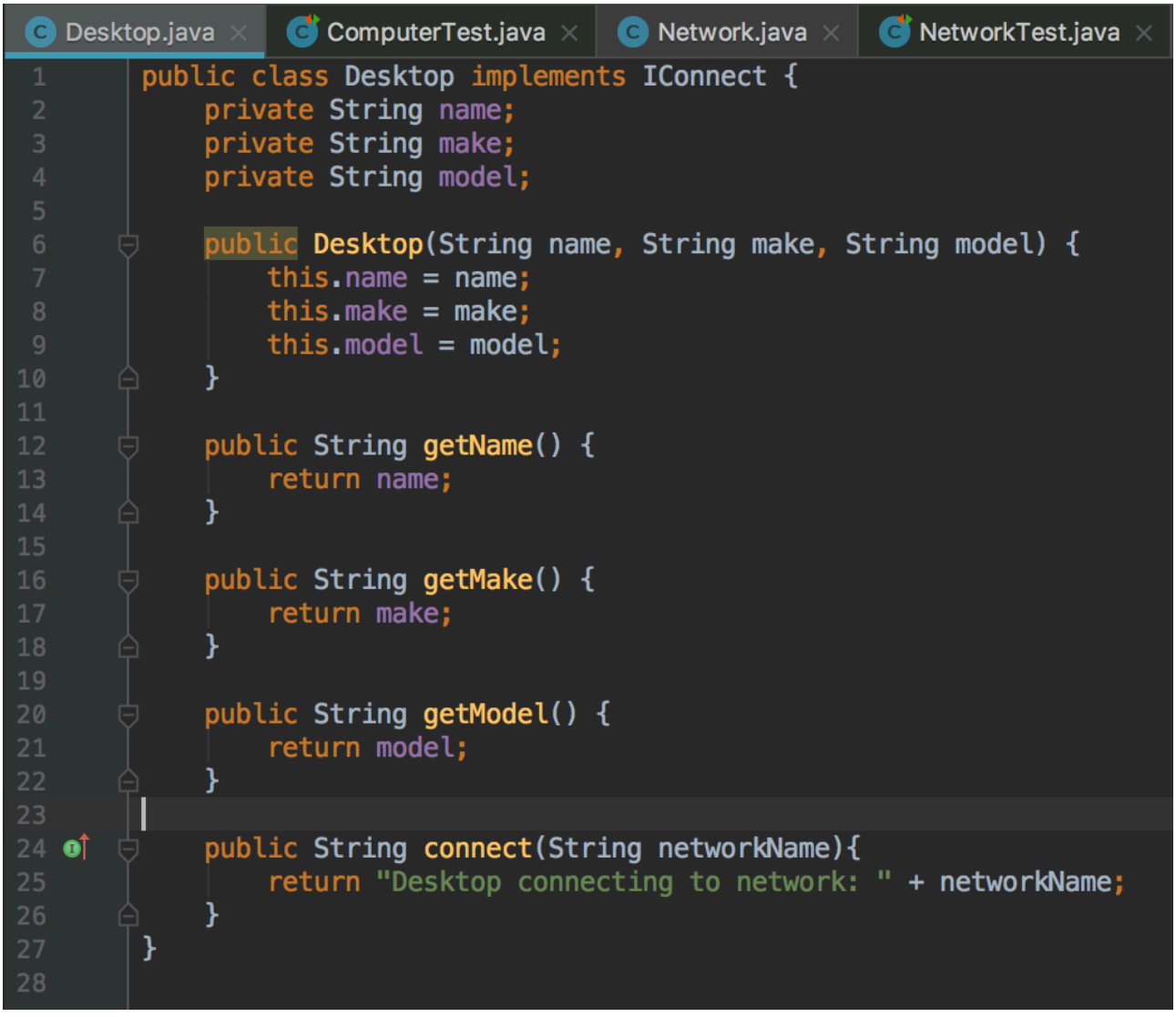
The result of the Customer.all function:

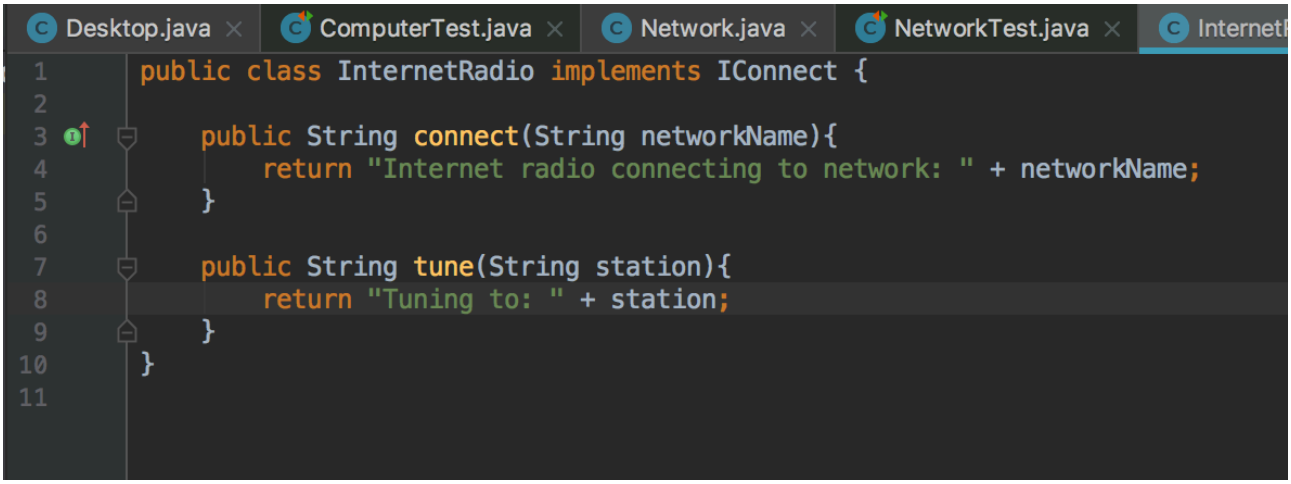
```
[➜  cinema git:(master) ✗ ruby console.rb                    ]

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:

Classes implementing the IConnect interface:




Desktop.java
ComputerTest.java
Network.java
NetworkTest.java
public class InternetRadio implements IConnect {
public String connect(String networkName){
return "Internet radio connecting to network: " + networkName;
}
public String tune(String station){
return "Tuning to: " + station;
}
}


Desktop.java
ComputerTest.java
Network.java
Printer.java
NetworkTest.java
InternetRadio.java
public class Printer implements IConnect {
public String print(String data) { return "printing: " + data; }
public String connect(String networkName) { return "Printer connecting to network: " + networkName; }
}

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