Evidence for Implementation and Testing Unit

Andrew Laughlin
Cohort E21

I.T 1: Evidence for use of encapsulation in a program:

```java
import Interface.ISell;

import java.util.Date;


public abstract class Item implements ISell {
    private double buyPrice;
    private double shippingPrice;
    private double currentMarketValue;
    private int purchaseYear;
    private boolean isForResale;
    private boolean isFavourite;

    public Item(int purchaseYear, double buyPrice, double shippingPrice){
        this.buyPrice = buyPrice;
        this.shippingPrice = shippingPrice;
        this.purchaseYear = purchaseYear;
        this.isForResale = false;
        this.isFavourite = false;
    }
```

```
public int getPurchaseYear() { return purchaseYear; }

public void setPurchaseYear(int newPurchaseYear) { this.purchaseYear = newPurchaseYear; }

public double getBuyPrice() { return buyPrice; }

public void setBuyPrice(double newBuyPrice) { this.buyPrice = newBuyPrice; }

public double getShippingPrice() { return shippingPrice; }

public void setShippingPrice(double newShippingPrice) {this.shippingPrice = newShippingPrice; }

public double getCurrentMarketValue() {
    return currentMarketValue;
}

public void setNewMarketValue(double newMarketValue) { this.currentMarketValue = newMarketValue; }

public boolean getFavouriteStatus(){
    return this.isFavourite;
}

public void changeFavouriteStatus() {
    this.isFavourite ^= true;
}

public boolean getResaleStatus() {
    return this.isForResale;
}
```

I.T 2: Evidence for use of inheritance in a program:

Screenshots showing:

- **A class**
- **A class that inherits from a previous class**
- **An object in the inherited class**
- **A method that uses the information inherited from another class**

```java
import Interface.ISell;

import java.util.Date;


public abstract class Item implements ISell {
    private double buyPrice;
    private double shippingPrice;
    private double currentMarketValue;
    private int purchaseYear;
    private boolean isForResale;
    private boolean isFavourite;

    public Item(int purchaseYear, double buyPrice, double shippingPrice){
        this.buyPrice = buyPrice;
        this.shippingPrice = shippingPrice;
        this.purchaseYear = purchaseYear;
        this.isForResale = false;
        this.isFavourite = false;
    }
```

```java
public class Drink extends Item {

    private String brand;
    private String model;
    private DrinkType drinkType;

    public Drink(String brand, String model, int purchaseYear, double buyPrice, double shippingPrice, DrinkType drinkType) {
        super(purchaseYear, buyPrice, shippingPrice);
        this.brand = brand;
        this.model = model;
        this.drinkType = drinkType;

    }
```

```java
public class DrinkTest {

    Drink drink;
    Date purchaseDate;

    @Before
    public void before() {

        purchaseDate = new Date();
        drink = new Drink( brand: "Smirnoff",  model: "Red Label", purchaseYear: 2012, buyPrice: 20.00, shippingPrice: 2.00, DrinkType.VODKA);
    }
```

```
@Test
public void canGetMake() { assertEquals( expected: "Smirnoff", drink.getBrand()); }

@Test
public void canGetModel() { assertEquals( expected: "Red Label", drink.getModel()); }


@Test
public void canGetPurchaseDate() { assertEquals( expected: 2012, drink.getPurchaseYear()); }

@Test
public void canGetBuyPrice() { assertEquals( expected: 20.00, drink.getBuyPrice(),  delta: 0.01); }

@Test
public void canGetShippingPrice() { assertEquals( expected: 2.00, drink.getShippingPrice(),  delta: 0.01); }
```

I.T 5: Evidence for use of an array in a program, a function that uses the array, and the result:

(Array of songs in a playlist, which the add_song method can be called on to add a song or find out if it is already in the playlist)

```ruby
    @guest1 = Guest.new("Andy", "Thunder Road", 20)
    @guest2 = Guest.new("Ali", "Animal Nitrate", 30)
    @guest3 = Guest.new("Eilidh", "Baa Baa Black Sheep", 10)

    @room1 = Room.new("Rock", [], 0, ["Disarm", "Home", "Animal Nitrate"], 15, 0)
    @room2 = Room.new("Pop", [@guest2, @guest3], 0, ["Wuthering Heights", "Night Fever", "Suspicious
    Minds"], 10, 10)

  end

  def test_get_name
    assert_equal("Rock", @room1.name)
  end

  def test_get_capacity
    assert_equal(0, @room1.capacity)
  end

  def test_get_guests
    assert_equal([], @room1.guests)
  end

  def test_get_playlist
    assert_equal(["Wuthering Heights", "Night Fever", "Suspicious Minds"], @room2.playlist)
  end

  def test_check_in_guest
    @room1.check_in_guest(@guest1)
    assert_equal("Andy", @guest1.name)
```

```ruby
  end

  def test_check_out_guest
    @room1.check_out_guest(@guest1)
    assert_equal([], @room1.guests)
  end

  def test_add_song_to_room
    @room2.add_song(@song1)
    assert_equal(4, @room2.playlist.length)
  end
```

```ruby
class Room

  attr_reader :name, :capacity, :entry_fee
  attr_accessor :guests, :playlist, :bar_tab

  def initialize(name, guests=[], capacity, playlist, entry_fee, bar_tab)
```

```ruby
  end

  def add_song(song)
    if @playlist.include?(song)
      return "That song is already in the playlist!"
    else
      @playlist << song
    end
  end
```

```
➜  ccc_homework git:(master) ruby specs/room_spec.rb
Run options: --seed 5819

# Running:

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

Finished in 0.002117s, 7557.8647 runs/s, 8030.2313 assertions/s.

16 runs, 17 assertions, 0 failures, 0 errors, 0 skips
➜  ccc_homework git:(master)
```

I.T 6: Evidence for use of a hash in a program, a function that uses the hash, and the result:

(Hash of drinks and a method which can be called on the hash to calculate the quantity of each drink)

```ruby
def setup
  @drink1 = Drink.new("JD", 5, 10)
  @drink2 = Drink.new("Beer", 3, 20)
  @drink3 = Drink.new("Wine", 4, 30)
  @drink4 = Drink.new("Vodka", 4, 40)

  @food1 = Food.new("Burger", 8, 15)

  @customer1 = Customer.new("Euan", 25, 100, 20)
  @customer2 = Customer.new("Andy", 16, 90, 20)

  # @pub = Pub.new("Chanter", 500, [@drink1, @drink2, @drink3, @drink4])
  @pub = Pub.new("Chanter", 500, {:@drink1 => 5, :@drink2 => 10, :@drink3 => 8, :@drink4
  => 12})

end
```

```ruby
def test_stock_count
  count_stock = @pub.stock_count()
  assert_equal(35, count_stock)
end
```

```ruby
def stock_count()
  stock = 0
  @drinks.each do |d, q|
    stock += q
  end
  return stock
end
```

```
➜  pub_river_drink git:(master) ✗ atom .
➜  pub_river_drink git:(master) ✗ ruby specs/pub_spec.rb
Run options: --seed 54348

# Running:

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

Finished in 0.001719s, 7562.5372 runs/s, 8144.2708 assertions/s.

13 runs, 14 assertions, 0 failures, 0 errors, 0 skips
➜  pub_river_drink git:(master) ✗
```

I.T 3: Evidence for the use of searching for data in a program:

(Function showing a function which searches for and displays all of the customers in the 'customers' table.)

```ruby
def self.all()
  sql = "SELECT * FROM customers"
  customer_hashes = SqlRunner.run(sql)
  return self.map_items(customer_hashes)
end
```

```
[2] pry(main)> !!!
[➜  weekend_CCC_homework git:(master) ✗ ruby db/console.rb

      /Users/user/codeclan_work/week_03/day_5/weekend_CCC_homework/db/console.rb @ 1

    129:   "film_id" => film2.id,
    130:   "show_time" => '15:45'
    131:   })
    132: screening7.save
    133:
 => 134:   binding.pry
    135:   nil

[1] pry(main)> Customer.all
=> [#<Customer:0x007ff50ecee7c0 @funds=30, @id=25, @name="Andy">,
 #<Customer:0x007ff50ecee6f8 @funds=50, @id=26, @name="Ali">,
 #<Customer:0x007ff50ecee630 @funds=80, @id=27, @name="Kirsty">,
 #<Customer:0x007ff50ecee568 @funds=100, @id=28, @name="Ed">]
[2] pry(main)>
```

```ruby
  end
```

```
ccc=# SELECT * FROM customers;
 id |  name  | funds
----+--------+-------
 25 | Andy   |    30
 26 | Ali    |    50
 27 | Kirsty |    80
 28 | Ed     |   100
(4 rows)
```

I.T 4 Evidence for the sorting of data in a program:

(function showing a method which calls for all of the films watched by a certain customer, by joining the ‘customer’ and ‘film’ tables together via the ‘tickets’ table.

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

```
[5] pry(main)>
[6] pry(main)> customer2.films
=> [#<Film:0x007ff50ec24e98 @id=25, @price=12, @title="The Godfather">,
 #<Film:0x007ff50ec24dd0 @id=27, @price=8, @title="The Good, The Bad and The Ugly">]
[7] pry(main)>
```

I.T 7

Use of polymorphism in a program:
Example shows a guitar being used as an individual class and also as an instrument in an array of instruments:

- The guitar constructor being created
- The shop class being created with an array of instruments as stock
- The shop test showing a new guitar being created as an instrument
- The tests showing a new instrument(a guitar) being added to the shop stock

```
import Enums.InstrumentType;
import Interfaces.IPlay;

import java.util.ArrayList;

public class Guitar extends Instrument implements IPlay {

    private GuitarString strings;
    private Fretboard fretboard;

    public Guitar(String make, String model, double buyingPrice, double sellingPrice, GuitarString strings, F
        super(make, model, buyingPrice, sellingPrice, instrumentType);
        this.strings = strings;
        this.fretboard = fretboard;
    }

    public Accessory getStrings() {
        return strings;
    }

    public Fretboard getFretboard() {
        return fretboard;

    }

    public void changeFretboard(Fretboard newFretboard) {
        this.fretboard = newFretboard;
    }
```

```java
import java.util.ArrayList;

public class Shop {

    private String name;
    private ArrayList<Instrument> instrumentStock;
    private ArrayList<Accessory> accessoryStock;

    public Shop(String name){
        this.name = name;
        this.instrumentStock = new ArrayList<>();
        this.accessoryStock = new ArrayList<>();
    }

    public String getName() {
        return name;
    }

    public int countInstrumentStock() {
        return instrumentStock.size();
    }

    public int countAccessoryStock() {
        return accessoryStock.size();
    }

    public void addInstrument(Instrument instrument) {
        instrumentStock.add(instrument);
```

```java
import Enums.InstrumentType;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class ShopTest {

    Shop shop;
    Instrument instrument1, instrument2;
    GuitarString guitarString,guitarString2;
    Fretboard fretboard;
    Accessory accessory;

    @Before
    public void before(){
        shop = new Shop( name: "Ray's Music Store");
        guitarString = new GuitarString( make: "Fender",  buyingPrice: 10.00,  sellingPrice: 15.00);
        fretboard = new Fretboard( make: "Fender",  buyingPrice: 40.00,  sellingPrice: 60.00);
        instrument1 = new Guitar( make: "Fender",  model: "Stratocaster",  buyingPrice: 200.00,  sellingPrice: 550.00, guitarS
        instrument2 = new Guitar( make: "Gibson",  model: "Les Paul",  buyingPrice: 300.00,  sellingPrice: 500.00, guitarStrin
        accessory = new Fretboard( make: "Buffalo",  buyingPrice: 100.00,  sellingPrice: 120.00);
    }

    @Test
    public void canGetName(){
        assertEquals( expected: "Ray's Music Store", shop.getName());
    }
```

ShopTest

```
    @Test
    public void canAddInstrumentToStock(){
        shop.addInstrument(instrument1);
        assertEquals( expected: 1, shop.countInstrumentStock());
    }

    @Test
    public void  canAddAccessoryToStock(){
        shop.addAccessory(accessory);
        assertEquals( expected: 1, shop.countAccessoryStock());
    }

    @Test
    public void canRemoveInstrumentFromStock(){
        shop.addInstrument(instrument1);
        shop.addInstrument(instrument2);
        shop.removeInstrument(instrument1);
        assertEquals( expected: 1, shop.countInstrumentStock());
    }

    @Test
```