Evidence for Implementation and Testing Unit

David Rawson
Cohort E18

I.T.1 - An example of encapsulation.

```
class Game {

    private String hand1;
    private String hand2;

    public Game(String hand1, String hand2) {
        this.hand1 = hand1;
        this.hand2 = hand2;
        }
```

I.T.2 - Use of inheritance.

```
package instruments;

import behaviours.ISellable;

public abstract class Instrument implements ISellable{

    protected String make;
    protected String model;
    protected String family;
    protected double buyPrice;
    protected double sellPrice;

    public Instrument(String make, String model, String family, double buyPrice, double
    sellPrice){
        this.make = make;
        this.model = model;
        this.family = family;
        this.buyPrice = buyPrice;
        this.sellPrice = sellPrice;
    }

    public double calculateMarkup(){
      return this.sellPrice - this.buyPrice;
    }
```

Instrument is an abstract parent class.

```java
package instruments;

import behaviours.IPlayable;

public class Organ extends Instrument implements IPlayable {

    private boolean pacTested;

    public Organ(String make, String model, String family, double buyPrice, double
    sellPrice, boolean pacTested) {
        super(make, model, family, buyPrice, sellPrice);
        this.pacTested = pacTested;
    }

    public String play() {
        return "Extremely Ray Charles sound";
    }

    public boolean getPACTested() {
        return this.pacTested;
    }
}
```

Organ inherits from Instrument.

```java
public class OrganTest {

    Organ organ;

    @Before
    public void before(){
        organ = new Organ("Hammond", "B3", "Keyboard", 1250.0, 1600.00, true);
    }
```

organ is an instance of an Organ.

```java
    @Test
    public void canCalculateMarkup(){
        assertEquals( expected: 350.0, organ.calculateMarkup() ,  delta: 0.0);
    }
```

OrganTest › canCalculateMarkup()

All 22 tests passed - 6ms

The calculateMarkup() method is inherited from the parent class. The test passes.

I.T.3 - Searching data.

```ruby
def find
  db = PG.connect({dbname: 'bounty_hunters', host: 'localhost'})
  sql = "SELECT * FROM bounty_hunters WHERE id=$1"
  values = [@id]
  db.prepare("find", sql)
  hunters = db.exec_prepared("find", values)
  db.close()
  return hunters.map {|hunter| Bounty.new(hunter)}
end
```

```ruby
bounty_hunter2 = Bounty.new({
  'name' => 'Jango Fett',
  'species'  => 'slug',
  'weapon'  => 'whip',
  'bounty_value' => '100'
})

bounty_hunter1.save()
bounty_hunter2.save()


bounty_hunter1.name = ("Trevor Fett")

bounty_hunter1.update

p bounty_hunter2.find
```

```
➜ day_2 ruby console.rb
[#<Bounty:0x007fdbd8425160 @id=13, @name="Jango Fett", @species="slug", @weapo
n="whip", @bounty_value=100>]
➜ day_2 atom console.rb
➜ day_2
```

I.T.4 - Sorting data.

```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)
  result = film_hashes.map{|film_hash| Film.new(film_hash)}
  return result
end
```

```ruby
p customer2.name
p customer2.films.count
```

```
.rb
"Dickie Attenborough"
2
➜ codeclan_cinema git:(master)
```

I.T.5 - Use of an array

```ruby
class Pub

    attr_reader :name , :till_value

    def initialize(name)
      @name = name
      @till_value = 0.00
      @drinks_in_pub = []

    end

    def add_to_till(payment)
      @till_value += payment
    end

    def add_drinks(drink)
      return @drinks_in_pub.push(drink)
    end
```

```ruby
def test_add_drinks
  drinks_array = @pub.add_drinks(@drink1)
  assert_equal(1, drinks_array.length)
end
```

```
Finished in 0.001647s, 4857.3171 runs/s, 4857.3171 assertions/s.

8 runs, 8 assertions, 0 failures, 0 errors, 0 skips
➜  pub git:(master)
```

The array is initialised, the test adds a drink, the array length has increased and the tests pass.

I.T.6 - Use of an hash.

```ruby
@person1 = {
  name: "Rick",
  age: 12,
  monies: 1,
  friends: ["Jay","Keith","Dave", "Val"],
  favourites: {
    tv_show: "Friends",
    things_to_eat: ["charcuterie"]
  }
```

```ruby
def return_favourite_tv_show(person)
  return person[:favourites][:tv_show]
end
```

```ruby
def test_return_favourite_tv_show
  result = return_favourite_tv_show(@person1)
  assert_equal("Friends", result)
end
```

```
Finished in 0.001559s, 5131.4946 runs/s, 5772.9314 assertions/s.

8 runs, 9 assertions, 0 failures, 0 errors, 0 skips
➜  specs git:(master) ×
```

Created the hash, the test uses the method to extract information from the hash. All tests pass.

I.T.7 - Polymorphism

```java
public class Shop {

    ArrayList<ISellable> stock;

    public Shop() {
        this.stock = new ArrayList<>();
    }

    public void addStockItem(ISellable item){
        this.stock.add(item);
    }
```

```java
public interface ISellable {
    double calculateMarkup();
}
```

```java
package instruments;

import behaviours.ISellable;

public abstract class Instrument implements ISellable{

    protected String make;
    protected String model;
    protected String family;
    protected double buyPrice;
    protected double sellPrice;

    public Instrument(String make, String model, String family, double buyPrice, double
    sellPrice){
        this.make = make;
        this.model = model;
        this.family = family;
        this.buyPrice = buyPrice;
        this.sellPrice = sellPrice;
    }
```

The Shop class implementing polymorphism. Its stock array can hold any item that implements the ISellable interface.