Evidence for Implementation and Testing Unit

Rachel Johnson
E19

I.T 1 - Example of encapsulation

```java
public abstract class Kaiju implements IActionsForKaiju {
    private String name;
    private int healthValue;
    private int attackValue;

    public Kaiju(String name, int healthValue, int attackValue) {
        this.name = name;
        this.healthValue = healthValue;
        this.attackValue = attackValue;
    }

    public String getName() { return this.name; }

    public int getHealthValue() { return this.healthValue; }

    public int getAttackValue() { return this.attackValue; }

    public void attack(Vehicle vehicle) { vehicle.reduceHealthValue(this.attackValue); }

    public void attack(Building building) {
        building.reduceHealthValue(this.attackValue);
    }

    public void reduceHealthValue(int number) {
        if (number > this.healthValue) {
            this.healthValue = 0;
        } else {
            this.healthValue -= number;
        }

    }
}
```

```java
public abstract class Vehicle implements IGettersForVehicleAndBuilding, IActionsForVehicle {
    private String type;
    private int healthValue;

    public Vehicle(String type, int healthValue) {
        this.type = type;
        this.healthValue = healthValue;
    }

    public String getType() { return this.type; }

    public int getHealthValue() { return this.healthValue; }

    public void setHealthValue(int number) { this.healthValue = number; }

    public void reduceHealthValue(int number) {
        if (number > this.healthValue) {
            this.healthValue = 0;
        } else {
            this.healthValue -= number;
        }
    }

    public void attackWithTearGas(Kaiju kaiju) {
        kaiju.reduceHealthValue( number: 20);
    }

    public void attackWithGrenades(Kaiju kaiju) { kaiju.reduceHealthValue( number: 40); }
}
```

I.T 2 - Example of inheritance

Screenshot of parent class (Instrument):

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

public abstract class Instrument extends Stock implements IPlay {

    protected String material;
    protected String colour;
    protected InstrumentType instrumentType;

    public Instrument(int uniqueID, double buyPrice, double sellPrice, String description, String material, String
            colour, InstrumentType instrumentType) {
        super(uniqueID, buyPrice, sellPrice, description);
        this.material = material;
        this.colour = colour;
        this.instrumentType = instrumentType;
    }

    public String getMaterial() { return this.material; }

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

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

    public abstract String playSound();

}
```

Screenshot of inheriting class (Guitar):

```
import Enums.GuitarType;
import Enums.InstrumentType;

public class Guitar extends Instrument {

    private GuitarType guitarType;
    private int numberOfStrings;

    public Guitar(int uniqueID, double buyPrice, double sellPrice, String description, String material, String colour,
                  GuitarType guitarType, int numberOfStrings) {
        super(uniqueID, buyPrice, sellPrice, description, material, colour, InstrumentType.STRING);
        this.guitarType = guitarType;
        this.numberOfStrings = numberOfStrings;
    }

    @Override
    public String playSound() { return "Strum strum"; }

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

    public int getNumberOfStrings() { return this.numberOfStrings; }
}
```

Screenshot of Guitar object and methods from the inherited class:

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

import static org.junit.Assert.assertEquals;

public class GuitarTest {

    private Guitar guitar;
    private GuitarType guitarType;

    @Before
    public void before() {
        guitar = new Guitar( uniqueID: 1, buyPrice: 350,  sellPrice: 490, description: "guitar", material: "maple",
                colour: "surf green", guitarType.ELECTRIC,  numberOfStrings: 6);
    }

    @Test
    public void canGetMaterial() {
        assertEquals( expected: "maple", guitar.getMaterial());
    }

    @Test
    public void canGetColour() {
        assertEquals( expected: "surf green", guitar.getColour());
    }

    @Test
    public void canGetType() {
        assertEquals(InstrumentType.STRING, guitar.getType());
    }

    @Test
    public void canGetSound() {
        assertEquals( expected: "Strum strum", guitar.playSound());
    }

    @Test
    public void canGetGuitarType() {
        assertEquals(GuitarType.ELECTRIC, guitar.getGuitarType());
    }
```

I.T 3 - Example of searching

Table to be searched

```
 id |                  title                  |       author
----+-----------------------------------------+--------------------
  1 | Brave New World                         | Aldous Huxley
  2 | Oryx and Crake                          | Margaret Atwood
  3 | The Timetraveller's Wife                | Audrey Niffenegger
  4 | Charlie and the Chocolate Factory       | Roald Dahl
  5 | The Girl with All the Gifts             | M R Carey
  6 | Chinese Cinderella                      | Adeline Yen Mah
  7 | Cloud Atlas                             | David Mitchell
  8 | A Clockwork Orange                      | Anthony Burgess
  9 | The Lion, the Witch and the Wardrobe | C S Lewis
 10 | 1984                                    | George Orwell
(10 rows)
```

Search function

Function searches within the table for a specified id number and returns the associated book

```ruby
require('pg')

class Book

  attr_reader :id, :title, :author

  def initialize(inputs)
    @id = inputs['id'].to_i if inputs['id']
    @title = inputs['title']
    @author = inputs['author']
  end

  def Book.find_by_id(id)
    db = PG.connect({dbname: "books", host: "localhost"})
    sql = "SELECT * FROM books WHERE id = $1;"
    values = [id]
    db.prepare("find", sql)
    result = db.exec_prepared("find", values)
    db.close()
    return result.map {|book| Book.new(book)}
  end

end

p Book.find_by_id(5)

```

Result

```
[➜  PDA git:(master) ✗ ruby IT3_evidence.rb                                                                    ]
 [#<Book:0x007fcc45148778 @id=5, @title="The Girl with All the Gifts", @author="M R Carey">]
 ➜  PDA git:(master) ✗
```

I.T 4 - Example of sorting

Table to be sorted is same as that used for I.T 3.

Function

Function sorts the table by title in ascending order.

```ruby
require('pg')

class Book

  attr_reader :id, :title, :author

  def initialize(inputs)
    @id = inputs['id'].to_i if inputs['id']
    @title = inputs['title']
    @author = inputs['author']
  end

  def Book.sort()
    db = PG.connect({dbname: "books", host: "localhost"})
    sql = "SELECT * FROM books ORDER BY title ASC;"
    db.prepare("sort", sql)
    result = db.exec_prepared("sort", [])
    db.close()
    return result.map {|book| Book.new(book)}
  end

end

Book.sort().each{|book| p book}
# Added .each{...} functionality above so output of sort function is easier to
# read in the terminal

```

Result

```
➜  PDA git:(master) ✗ ruby IT4_evidence.rb
#<Book:0x007ffc910891d0 @id=10, @title="1984", @author="George Orwell">
#<Book:0x007ffc91089108 @id=8, @title="A Clockwork Orange", @author="Anthony Burgess">
#<Book:0x007ffc91088f28 @id=1, @title="Brave New World", @author="Aldous Huxley">
#<Book:0x007ffc91088ac8 @id=4, @title="Charlie and the Chocolate Factory", @author="Roald Dahl">
#<Book:0x007ffc91088938 @id=6, @title="Chinese Cinderella", @author="Adeline Yen Mah">
#<Book:0x007ffc910887f8 @id=7, @title="Cloud Atlas", @author="David Mitchell">
#<Book:0x007ffc91088708 @id=2, @title="Oryx and Crake", @author="Margaret Atwood">
#<Book:0x007ffc910882a8 @id=5, @title="The Girl with All the Gifts", @author="M R Carey">
#<Book:0x007ffc91088050 @id=9, @title="The Lion, the Witch and the Wardrobe", @author="C S Lewis">
#<Book:0x007ffc9108a918 @id=3, @title="The Timetraveller's Wife", @author="Audrey Niffenegger">
➜  PDA git:(master) ✗
```

I.T 5 - Example of an array, a function that uses an array and the result

Array and function

```ruby
cheesecake_ingredients = ["biscuits", "butter", "cream cheese", "sugar", "double cream", "raspberries"]

def display_ingredients(ingredients)
  p "The ingredients in this dish are:"
  ingredients. each {|ingredient| p ingredient}
end

display_ingredients(cheesecake_ingredients)

```

Result

```
➜  PDA git:(master) ✗ ruby IT5_evidence.rb
"The ingredients in this dish are:"
"biscuits"
"butter"
"cream cheese"
"sugar"
"double cream"
"raspberries"
➜  PDA git:(master) ✗
```

I.T 6 Example of a hash, a function that uses the hash and the result

Hash and function

```ruby
menu_prices = {
  starter: 6.50,
  main: 17.50,
  dessert: 5.50
}

def total_cost(menu)
running_total = 0
menu.each {|key, value| running_total += value}
p "The total cost of the meal is £#{ '%.2f' % running_total}"
end

total_cost(menu_prices)

```

Result

```
➜  PDA git:(master) ✗ ruby IT6_evidence.rb
"The total cost of the meal is £29.50"
➜  PDA git:(master) ✗
```

I.T 7 Example of polymorphism in a program

```java
package Interfaces;

public interface ISell {

    double markUp();

    int getUniqueID();
}
```

```java
import Interfaces.ISell;

public abstract class Stock implements ISell {

    protected double buyPrice;
    protected double sellPrice;
    protected String description;
    private int uniqueID;

    public Stock(int uniqueID, double buyPrice, double sellPrice, String description) {
        this.uniqueID = uniqueID;
        this.buyPrice = buyPrice;
        this.sellPrice = sellPrice;
        this.description = description;
    }

    public int getUniqueID() {
        return this.uniqueID;
    }

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

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

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

    public String getDescription() {
        return this.description;
    }
}
```

```java
import Enums.InstrumentType;

public abstract class Instrument extends Stock {

    protected String material;
    protected String colour;
    protected InstrumentType instrumentType;

    public Instrument(int uniqueID, double buyPrice, double sellPrice, String description, String material, String
            colour, InstrumentType instrumentType) {
        super(uniqueID, buyPrice, sellPrice, description);
        this.material = material;
        this.colour = colour;
        this.instrumentType = instrumentType;
    }

    public String getMaterial() { return this.material; }

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

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

    public abstract String playSound();

}
```

```java
import Enums.InstrumentType;

public class Flute extends Instrument {

    public Flute(int uniqueID, double buyPrice, double sellPrice, String description, String material, String colour) {
        super(uniqueID, buyPrice, sellPrice, description, material, colour, InstrumentType.WOODWIND);
    }

    @Override
    public String playSound() {
        return "Toot toot!";
    }
}
```

```java
public class Metronome extends Stock {

    public Metronome(int uniqueID, double buyPrice, double sellPrice, String description) {
        super(uniqueID, buyPrice, sellPrice, description);
    }
}
```

```java
import Interfaces.ISell;

import java.util.ArrayList;

public class Shop {

    private ArrayList<ISell> stock;

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

    public ArrayList<ISell> getStock() { return this.stock; }

        public void addToStock(ISell stockItem, int quantity) {
            for (int i = 1; i <= quantity; i++) {
                this.stock.add(stockItem);
            }
        }

        public boolean stockContainsID(int id) {
            for (ISell item: this.stock) {
                if (item.getUniqueID() == id) {
                    return true;
                }
            }
            return false;
        }

        public ISell findByID(int id) {
            ISell found = null;
            for (ISell item: this.stock) {
                if (item.getUniqueID() == id) {
                    found = item;
                }
            }
            return found;
        }

        public void removeFromStock(ISell stockItem, int quantity) {
            for (int i = 1; i <= quantity; i++) {
                if (stockContainsID(stockItem.getUniqueID())) {
                    ISell itemToBeRemoved = findByID(stockItem.getUniqueID());
                    this.stock.remove(itemToBeRemoved);
                    }
                }
            }
```

```java
        public double totalPotentialProfit() {
            double total = 0;
            for (ISell item: this.stock) {
                total += item.markUp();
            }
            return total;
        }
}
```

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

import java.util.ArrayList;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

public class ShopTest {
    private Shop shop;
    private Flute flute;
    private Metronome metronome;

    @Before
    public void before() {
        shop = new Shop();
        flute = new Flute( uniqueID: 1, buyPrice: 120,  sellPrice: 190, description: "flute",  material: "silver",  colour: "metallic");
        metronome = new Metronome( uniqueID: 2, buyPrice: 4.50,  sellPrice: 10.50,  description: "metronome");
    }


    @Test
    public void canGetStockLevelsWhenEmpty() { assertEquals(new ArrayList<>(), shop.getStock()); }

    @Test
    public void canAddToStock() {
        assertEquals( expected: 0,shop.getStock().size());
        assertFalse(shop.getStock().contains(flute));
        assertFalse(shop.getStock().contains(metronome));
        shop.addToStock(flute,  quantity: 4);
        assertEquals( expected: 4, shop.getStock().size());
        assertTrue(shop.getStock().contains(flute));
        assertFalse(shop.getStock().contains(metronome));
        shop.addToStock(metronome,  quantity: 10);
        assertEquals( expected: 14, shop.getStock().size());
        assertTrue(shop.getStock().contains(flute));
        assertTrue(shop.getStock().contains(metronome));
    }

    @Test
    public void canCheckIfStockContainsID() {
        shop.addToStock(flute, quantity: 4);
        assertTrue(shop.stockContainsID(1));
        assertFalse(shop.stockContainsID(2));
    }
```

```java
    @Test
    public void canFindByID() {
        shop.addToStock(flute, quantity: 4);
        shop.addToStock(metronome, quantity: 5);
        assertEquals(flute, shop.findByID(1));
        assertEquals(metronome, shop.findByID(2));
    }

    @Test
    public void canRemoveFromStock() {
        shop.addToStock(flute, quantity: 3);
        shop.addToStock(metronome, quantity: 8);
        shop.removeFromStock(flute, quantity: 2);
        assertEquals( expected: 9, shop.getStock().size());
        shop.removeFromStock(flute, quantity: 2);
        assertEquals( expected: 8, shop.getStock().size());
    }

    @Test
    public void canGetTotalPotentialProfit() {
        shop.addToStock(flute,  quantity: 4);
        shop.addToStock(metronome, quantity: 8);
        assertEquals( expected: 328, shop.totalPotentialProfit(),  delta: 0.01);
    }
}
```