Evidence for Implementation and Testing Unit.

Bert Overduin
Cohort E19

I.T 1 - Demonstrate one example of encapsulation that you have written in a program.

Item.java

```java
package ShoppingBasket;

public class Item {

    private String name;
    private double price;
    private int quantity;
    private Boolean bogof;

    public Item (String name, double price, int quantity, Boolean bogof) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
        this.bogof = bogof;
    }

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

    public double getPrice() {
        return this.price;
    }

    public int getQuantity() {
        return this.quantity;
    }

    public Boolean hasBogof() {
        return this.bogof;
    }

}
```

I.T 2 - Example of the use of inheritance in a program.

Room Class:

Room.java

```java
package Hotel.Rooms;

import Hotel.Guest;

import java.util.ArrayList;

public abstract class Room {

    private int number;
    private int capacity;
    private ArrayList<Guest> guests;

    public Room(int number, int capacity) {
        this.number = number;
        this.capacity = capacity;
        this.guests = new ArrayList<>();
    }

    public int getNumber() {
        return this.number;
    }

    public int getCapacity() {
        return this.capacity;
    }

    public int countGuests() {
        return this.guests.size();
    }

    public void add(Guest guest) {
        this.guests.add(guest);
    }

    public void remove(Guest guest) {
        this.guests.remove(guest);
    }

    public ArrayList<Guest> getGuests() {
        return this.guests;
    }

    public boolean isFull() {
        return this.guests.size() >= this.capacity;
    }
```

The **BedRoom** Class inherits from the **Room** Class:

The **isVacant()** method uses the **guests** attribute and **getGuests()** method, which are inherited from the **Room** Class:

BedRoom.java

```java
package Hotel.Rooms;

import Hotel.Guest;

import java.util.ArrayList;

public class BedRoom extends Room {

    private double rate;
    private Type type;

    public BedRoom(int number, int capacity, double rate, Type type) {
        super(number, capacity);
        this.rate = rate;
        this.type = type;
    }

    public double getRate() {
        return this.rate;
    }

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

    public boolean isVacant() {
        ArrayList<Guest> guests = getGuests();
        return guests.size() == 0;
    }

}
```

An **Object** in the **BedRoom** Class:

BedRoomTest.java

```java
import Hotel.Guest;
import Hotel.Rooms.BedRoom;
import Hotel.Rooms.Type;

import org.junit.Before;
import org.junit.Test;

import java.util.Arrays;

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

public class BedRoomTest {

    BedRoom bedroom;
    Guest guest1, guest2;

    @Before
    public void before() {
        bedroom = new BedRoom( number: 1,  capacity: 1,  rate: 100.00, Type.SINGLE);
        guest1 = new Guest( name: "Stuart Hogg");
        guest2 = new Guest( name: "Finn Russell");
    }

    @Test
    public void hasNumber() {
        assertEquals( expected: 1, bedroom.getNumber());
    }

    @Test
    public void hasCapacity() {
        assertEquals( expected: 1, bedroom.getCapacity());
    }

    @Test
    public void canAddGuest() {
        bedroom.add(guest1);
        assertEquals( expected: 1, bedroom.countGuests());
    }

    @Test
    public void canRemoveGuest() {
        bedroom.add(guest1);
        bedroom.remove(guest1);
        assertEquals( expected: 0, bedroom.countGuests());
    }

    @Test
    public void canGetGuests() {
        bedroom.add(guest1);
        bedroom.add(guest2);
        assertEquals(Arrays.asList(guest1, guest2), bedroom.getGuests());
    }

    @Test
    public void checkRoomFull() {
        bedroom.add(guest1);
        assertTrue(bedroom.isFull());
    }

    @Test
    public void checkRoomNotFull() {
        assertFalse(bedroom.isFull());
    }

    @Test
    public void checkBedRoomVacant() {
        assertTrue(bedroom.isVacant());
    }

    @Test
    public void checkBedRoomNotVacant() {
        bedroom.add(guest1);
        assertFalse(bedroom.isVacant());
    }

    @Test
    public void hasRate() {
        assertEquals( expected: 100.00, bedroom.getRate(),  delta: 0.01);
    }

    @Test
    public void hasType() {
        assertEquals(Type.SINGLE, bedroom.getType());
    }

}
```

I.T 3 - Example of searching

```ruby
benelux = ['The Netherlands', 'Belgium', 'Luxembourg']

def search_word(list, searched_word)
  for word in list
    return "#{searched_word} found" if word == searched_word
  end
  return "#{searched_word} not found"
end

puts search_word(benelux, "The Netherlands")

puts search_word(benelux, "Italy")
```

```
➜  code git:(master) ✗ ruby searching.rb
The Netherlands found
Italy not found
```

I.T 4 - Example of sorting.

```ruby
benelux = ['The Netherlands', 'Belgium', 'Luxembourg']

def sort_words(unsorted)
  still_unsorted = unsorted
  sorted = []
  while still_unsorted.length > 1
    unsorted = still_unsorted
    still_unsorted = []
    smallest = unsorted.pop
    for word in unsorted
      if word < smallest
        still_unsorted << smallest
        smallest = word
      else
        still_unsorted << word
      end
    end
    sorted << smallest
  end
  sorted << unsorted[0]
end

puts sort_words(benelux)
```

```
➜  code git:(master) ✗ ruby sorting.rb
Belgium
Luxembourg
The Netherlands
```

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

```ruby
benelux = ['The Netherlands', 'Belgium', 'Luxembourg']

def count_characters(list)
  for word in list
    puts "#{word} contains #{word.length()} characters"
  end
end

count_characters(benelux)
```

```
➜  code git:(master) ✗ ruby array.rb
The Netherlands contains 15 characters
Belgium contains 7 characters
Luxembourg contains 10 characters
```

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

```ruby
benelux = [
  {
    name: "The Netherlands",
    population: 17200671,
    capital: "Amsterdam"
  },
  {
    name: "Belgium",
    population: 11358357,
    capital: "Brussels"
  },
  {
    name: "Luxembourg",
    population: 590667,
    capital: "Luxembourg City"
  }
]

def calc_total_population(countries)
  total_population = 0
  for country in countries
    total_population += country[:population]
  end
  return total_population
end

puts calc_total_population(benelux)
```

```
➜  code git:(master) ✗ ruby hash.rb
29149695
```

I.T 7 - Example of polymorphism in a program.

Shop.java

```java
package Items;

import Behaviours.ISell;

import java.util.ArrayList;

public class Shop {

    private ArrayList<ISell> stock;

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

    public void addItemToStock(ISell item) {
        this.stock.add(item);
    }

    public void removeItemFromStock(ISell item) {
        this.stock.remove(item);
    }

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

    public double calculateMaxProfit() {
        double maxProfit = 0.00;
        for (ISell item: stock) {
            maxProfit += item.calculateMarkup();
        }
        return maxProfit;
    }

}
```

ISell.java

```java
package Behaviours;

public interface ISell {

    public double calculateMarkup();

}
```

IPlay.java

```java
package Behaviours;

public interface IPlay {

    public String play();

}
```

Item.java

```java
package Items;

import Behaviours.ISell;

public abstract class Item implements ISell {
    private String description;
    private double buyingPrice;
    private double sellingPrice;

    public Item (String description, double buyingPrice, double sellingPrice) {
        this.description = description;
        this.buyingPrice = buyingPrice;
        this.sellingPrice = sellingPrice;
    }

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

    public double getBuyingPrice() {
        return this.buyingPrice;
    }

    public double getSellingPrice() {
        return this.sellingPrice;
    }

    public double calculateMarkup() {
        return getSellingPrice() - getBuyingPrice();
    }

}
```

Instrument.java

```java
package Items.Instruments;

import Items.Item;

public abstract class Instrument extends Item {

    private Type type;

    public Instrument(Type type, String description, double buyingPrice, double sellingPrice) {
        super(description, buyingPrice, sellingPrice);
        this.type = type;
    }

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

}
```

Cello.java

```java
package Items.Instruments;

import Behaviours.IPlay;

public class Cello extends Instrument implements IPlay {

    private String builder;

    public Cello(Type type, String description, String builder, double buyingPrice, double sellingPrice) {
        super(type, description, buyingPrice, sellingPrice);
        this.builder = builder;
    }

    public String getBuilder() {
        return this.builder;
    }

    public String play() {
        return "The sound of the cello ...";
    }

}
```

Viola.java

```java
package Items.Instruments;

import Behaviours.IPlay;

public class Viola extends Instrument implements IPlay {

    private String builder;

    public Viola(Type type, String description, String builder, double buyingPrice, double sellingPrice) {
        super(type, description, buyingPrice, sellingPrice);
        this.builder = builder;
    }

    public String getBuilder() {
        return this.builder;
    }

    public String play() {
        return "The sound of the viola ...";
    }

}
```