Implementation and Testing Unit – Evidence

David Ellis
Cohort E18

I.T. 1 – Encapsulation Example Screenshot

```java
public class Card {

    private Suit suit;
    private Rank rank;

    public Card(Suit suit, Rank rank){
        this.suit = suit;
        this.rank = rank;
    }

    public Suit getSuit() {
        return this.suit;
    }

    public Rank getRank() {
        return this.rank;
    }
}
```

I.T. 2 – Inheritance Example Screenshots

A class:

```java
package staff;

public class Employee {

    private String name;
    private String ni;
    protected double salary;
    protected double taxPercentage;

    public Employee(String name, String ni, double salary, double taxPercentage){
        this.name = name;
        this.ni = ni;
        this.salary = salary;
        this.taxPercentage = taxPercentage;
    }

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

    public String getNi() {
        return this.ni;
    }

    public double getSalary() {
        return this.salary;
    }

    public double getTaxPercentage() {
        return this.taxPercentage;
    }
```

A class that inherits from the previous class:

```java
package management;

import staff.Employee;

public class Manager extends Employee {

    private String deptName;

    public Manager(String name, String ni, double salary, double taxPercentage, String deptName){
        super(name, ni, salary, taxPercentage);
        this.deptName = deptName;
    }

    public double calculateTaxAmount() {
        return (this.salary * (this.taxPercentage/100));
    }

    public String getDeptName() {
        return deptName;
    }

}
```

An object in the inherited class:

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

import static org.junit.Assert.assertEquals;

public class ManagerTest {

    Manager manager;

    @Before
    public void before(){
        manager = new Manager( name: "Joe",  ni: "AAA",  salary: 50000.00,  taxPercentage: 20.00,  deptName: "Finance");
    }

    @Test
    public void hasName(){
        assertEquals( expected: "Joe", manager.getName());
    }

    @Test
    public void hasNi(){
        assertEquals( expected: "AAA", manager.getNi());
    }

    @Test
    public void hasSalary(){
        assertEquals( expected: 50000.00, manager.getSalary(),  delta: 0.01);
    }

    @Test
    public void canCalculateTaxAmount(){
        assertEquals( expected: 10000.00, manager.calculateTaxAmount(),  delta: 0.01);
    }

    @Test
    public void getDeptName(){
        assertEquals( expected: "Finance", manager.getDeptName());
    }
}
```

A method that uses the information inherited from another class:

```java
    public Manager(String name, String ni, double salary, double taxPercentage, String deptName){
        super(name, ni, salary, taxPercentage);
        this.deptName = deptName;
    }

    public double calculateTaxAmount() {
        return (this.salary * (this.taxPercentage/100));
    }
```

I.T. 3 – Searching Data Example Screenshots

The searchForString method searches for a specific string in an ArrayList of String objects and returns true if it finds the string or false if it does not find the string.

```java
package IT.IT3;

import java.util.ArrayList;

public class SearchMethod {

    public static boolean SearchForString(String searchString, ArrayList<String> strings){

        boolean stringFound = false;

        for (int i = 0; i < strings.size(); i++){
            if (strings.get(i) == searchString){
                stringFound = true;
            }
        }
        return stringFound;
    }
}
```

Result of calling SearchForString method:

```java
    public static void main(String[] args) {
        String string1 = "Hello World";
        String string2 = "Java is great";
        String string3 = "String";
        String string4 = "12345";
        String string5 = "Lorem Ipsum";
        ArrayList<String> strings = new ArrayList<>();
        strings.add(string1);
        strings.add(string2);
        strings.add(string3);
        strings.add(string4);
        strings.add(string5);

        boolean result;
        result = SearchMethod.SearchForString( searchString: "Hello World", strings);

        System.out.println("Searching for 'Hello World'");
        System.out.println("Result: " + result);
        System.out.println();

        result = SearchMethod.SearchForString( searchString: "abcde", strings);
        System.out.println("Searching for 'abcde'");
        System.out.println("Result: " + result);

    }
}
```

```
Run  Runner (1)
/Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/bin/java ...
Searching for 'Hello World'
Result: true

Searching for 'abcde'
Result: false

Process finished with exit code 0
```

I.T. 4 – Sorting Data Example Screenshots

The sortDice method sorts an ArrayList of Die objects (a Die object has a value parameter) by using a comparator to tell Collections.sort that it needs to sort the collection of dice based on the die value.

```java
package IT.IT4;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class SortMethod {

    public static void sortDice(ArrayList<Die> dice){

        Collections.sort(dice, new Comparator<Die>() {

            public int compare(Die die1, Die die2) {
                return die1.getValue() - die2.getValue();
            }
        });

    }
}
```

Result of calling the sortDice method:

```java
        System.out.println("Unsorted dice:");

        for (int i = 0; i < dice.size(); i++){
            System.out.println("die value = " + dice.get(i).getValue());
        }
        System.out.println();
        System.out.println("Sorted dice:");

        SortMethod.sortDice(dice);

        for (int i = 0; i < dice.size(); i++){
            System.out.println("die value = " + dice.get(i).getValue());
        }
```

Run Runner

```
Unsorted dice:
die value = 1
die value = 6
die value = 4
die value = 2
die value = 3

Sorted dice:
die value = 1
die value = 2
die value = 3
die value = 4
die value = 6
```

I.T. 5 – Array Example Screenshots

An array of integers containing 8 elements is created in the Runner class below

```java
package IT.IT5;

public class Runner {

    public static void main(String[] args) {

        int[] intArray = {1,4,83,0,99,123,10,100};

        UsingArrays.printArrayContents(intArray);

    }
}
```

The printArrayContents method shown below takes in an array of integers and prints out the contents of the array

```java
package IT.IT5;

public class UsingArrays {

    public static void printArrayContents(int[] intArray){

        for (int i = 0; i < intArray.length; i++){
            System.out.println(intArray[i]);
        }
    }
}
```

The output from running the Runner class is shown below, with the contents of the array of integers printed out

```
Run  Runner (2)
/Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/bin/java ...
1
4
83
0
99
123
10
100

Process finished with exit code 0
```

I.T. 6 – Hash Example Screenshots

A hashmap of integers and strings containing 6 entries is created in the Runner class below

```java
package IT.IT6;

import java.util.HashMap;

public class Runner {

    public static void main(String[] args) {

        HashMap<Integer, String> hashMap = new HashMap<>();
        hashMap.put(1, "Entry 1");
        hashMap.put(2, "Entry 2");
        hashMap.put(3, "Entry 3");
        hashMap.put(4, "Entry 4");
        hashMap.put(5, "Entry 5");
        hashMap.put(6, "Entry 6");

        UsingHashes.printHashContents(hashMap);
    }
}
```

The printHashContents method shown below takes in a hashmap of integers and strings and prints out the contents of the hashmap

```java
package IT.IT6;

import java.util.HashMap;

public class UsingHashes {

    public static void printHashContents(HashMap<Integer, String> hashMap){

        for (int i = 1; i <= hashMap.size(); i++){
            System.out.println(hashMap.get(i));
        }
    }
}
```

The output from running the Runner class is shown below, with the string values of the hashmap printed out

```
Run  Runner (3)
/Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/bin/java ...
Entry 1
Entry 2
Entry 3
Entry 4
Entry 5
Entry 6

Process finished with exit code 0
```

I.T. 7 – Polymorphism Example Screenshots

MusicShop class which has an ArrayList of ISellable item objects:

```java
package MusicShop;

import MusicShop.Behaviours.ISellable;
import MusicShop.Items.Item;

import java.util.ArrayList;

public class Shop {

    private String name;
    private ArrayList<ISellable> items;

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

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

    public int getNumberOfItems() {
        return this.items.size();
    }

    public void addItem(Item item) {
        this.items.add(item);
    }

    public void removeItem(Item item) {
        this.items.remove(item);
    }
```

Item class, which implements the **ISellable** interface:

```java
package MusicShop.Items;

import MusicShop.Behaviours.ISellable;

public abstract class Item implements ISellable {

    private double buyPrice;
    private double sellPrice;

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

Accessory class, which extends the **Item** class (and therefore implements the **ISellable** interface):

```java
package MusicShop.Items.Accessories;

import MusicShop.Items.Item;

public class Accessory extends Item {

    private String accessoryType;
    private String description;

    public Accessory(double buyPrice, double sellPrice, String accessoryType, String description){
        super(buyPrice, sellPrice);
        this.accessoryType = accessoryType;
        this.description = description;
    }

    public String getAccessoryType() {
        return this.accessoryType;
    }

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

Instrument class, which extends the **Item** class (and therefore implements the **ISellable** interface):

```java
package MusicShop.Items.Instruments;

import ...

public abstract class Instrument extends Item {

    private InstrumentType instrumentType;
    private String material;
    private String colour;

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

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

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

Guitar class, which extends the **Instrument** class (and therefore implements the **ISellable** interface):

```java
package MusicShop.Items.Instruments;

import ...

public class Guitar extends Instrument implements IPlayable {

    private int numberOfStrings;
    private GuitarType guitarType;

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

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

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

    public String play() { return "Strumming on a guitar"; }

}
```

ISellable interface:

```java
package MusicShop.Behaviours;

public interface ISellable {

    double calculateMarkup();
}
```