Gergo Nagy - Implementation and Testing Unit (SQA PDA: Software Development)

I.T 1 Encapsulation in a program

```
package driver_management;
import behaviours.*;

public class Driver {
  private String name;
  private Driveable drive;

  public Driver(String name, Driveable drive){
    this.name = name;
    this.drive = drive;
  }

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

  public Driveable getDrive(){
    return this.drive;
  }

  public int driveTime(int distance){
    return this.drive.driveTime(distance);
  }

}
```

I.T 2 Inheritance in a program

```
public class PianoTest {

  Piano piano;

  @Before
  public void before(){
    piano = new Piano("Wood", "Black", "Classic", 500, 690, 80);
  }

  @Test
  public void hasKeys(){
    assertEquals(80, piano.getKeys());
  }

  @Test
  public void canPlay(){
    assertEquals("Dandadadaannn", piano.play());
  }

  @Test
  public void hasMaterial(){
    assertEquals("Wood", piano.getMaterial());
  }

  @Test
  public void hasColor(){
    assertEquals("Black", piano.getColor());
  }

  @Test
  public void hasType(){
    assertEquals("Classic", piano.getType());
  }

  @Test
  public void buyingPrice(){
    assertEquals(500, piano.getBuyPrice());
  }
```

```
package instrument_management;

public abstract class Instrument {

  String material;
  String color;
  String type;
  int buyPrice;
  int sellPrice;

  public Instrument(String material, String color, String type, int buyPrice, int sellPrice){
    this.material = material;
    this.color = color;
    this.type = type;
    this.buyPrice = buyPrice;
    this.sellPrice = sellPrice;
  }

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

  public String getColor(){
    return this.color;
  }

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

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

  public int getSellPrice(){
    return this.sellPrice;
  }
}
```

```java
package instrument_management;
import behaviours.*;

public class Piano extends Instrument implements Playable, Sellable{

  int numberOfKeys;

  public Piano(String material, String color, String type, int buyPrice, int sellPrice, int numberOfKeys){
    super(material, color, type, buyPrice, sellPrice);
    this.numberOfKeys = numberOfKeys;
  }

  public int getKeys(){
    return this.numberOfKeys;
  }

  public String play(){
    return "Dandadadaannn";
  }

  public int calculateMarkup(){
    return sellPrice - buyPrice;
  }
}
```

I.T 3 Demonstrate searching data in a program:

```ruby
  array = ["apple", "snake", "time", "please"]


def search(array, input)
  results = array.select {|array| array.include? input}
end


  results = search(array, "ple")

  puts  results
```

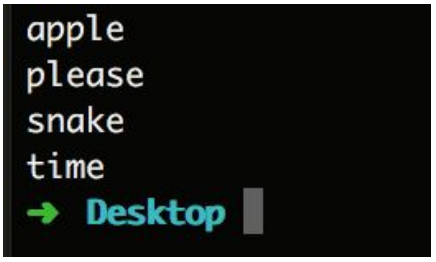
Result:

```
➜ Desktop ruby r.rb
apple
please
➜ Desktop
```

I.T 4 Demonstrate searching and sorting data in a program:

```ruby
array = ["apple", "snake", "time", "please"]


def search(array, input)
  results = array.select {|array| array.include? input}
end

def sort_an_array(array)
  array.sort!
end


  results = search(array, "ple")
  result2 = sort_an_array(array)

  puts result2
```

Result:



I.T 5 An array in a program:

```javascript
describe('Array tasks', function () {

    it('should concatenate two arrays, returning a new array', function () {
        var arr1 = [1, 2, 3]
        var arr2 = [4, 5, 6]
        var expectation = [1, 2, 3, 4, 5, 6]
        assert.deepEqual(arrayTasks.concat(arr1, arr2), expectation)
    })

    it('should insert an item in an array at any index position', function () {
        var arr = [1, 2, 4]
        assert.deepEqual(arrayTasks.insertAt(arr, 3, 2), [1, 2, 3, 4])
    })

    it('should square all values in an array, returning a new array', function () {
        var arr = [1, 2, 3, 4, 5]
        assert.deepEqual(arrayTasks.square(arr), [1, 4, 9, 16, 25])
    })

    it('should calculate the sum of all values in an array', function () {
        var arr = [1, 2, 3, 4, 5]
        assert.equal(arrayTasks.sum(arr), 15)
    })

    it('should find duplicate values in an array, returning a new array of the duplicates', function () {
        var arr = [1, 2, 3, 4, 4, 5, 5, 5]
        assert.deepEqual(arrayTasks.findDuplicates(arr), [4, 5])
    })
```

A function that uses the array:

```
var arrayTasks = {

    concat: function (arr1, arr2) {
        return arr1.concat(arr2);
    },

    insertAt: function (arr, itemToAdd, index) {
         arr.splice(index, 0, itemToAdd);
         return arr;
    },

    square: function (arr) {
        var newArr = [];
        arr.forEach(function(num) {
            newArr.push(num * num);
        })
        return arr = newArr;
    },

    sum: function (arr) {
         return sum = arr.reduce(function(a, b){
            return a + b;
        },0)
    },

    findDuplicates: function (arr) {
        var result = [];

        arr.forEach(function(number, index){
            if(arr.indexOf(number, index + 1) > -1){
                if(result.indexOf(number) === -1){
                    result.push(number);
                }
            }
        });
```

The result of the function:

```
Array tasks
  ✓ should concatenate two arrays, returning a new array
  ✓ should insert an item in an array at any index position
  ✓ should square all values in an array, returning a new array
  ✓ should calculate the sum of all values in an array
  ✓ should find duplicate values in an array, returning a new array of the duplicates
  ✓ should remove all instances of a value from an array, returning a new array
  ✓ should find all occurences of a value, returning an array of index positions
  ✓ should calculate the sum of all of even numbers in an array squared


8 passing (8ms)
```

I.T 6

Add data to the new Hash, display count, delete a key and value:

```ruby
abcd = Hash.new
abcd[100] = "a"
abcd[200] = "b"
abcd[300] = "c"
abcd[400] = "d"
abcd[500] = "e"



def count_hash(abcd)
  abcd.count
end

def delete(abcd, num)
  abcd.delete(100)
end


count = count_hash(abcd)
puts count

delete = delete(abcd, 100)


puts count1
```

Display the result:

I.T 7 Demonstrate the use of Polymorphism in a program:

```java
package instrument_management;

public abstract class Instrument {

  String material;
  String color;
  String type;
  int buyPrice;
  int sellPrice;

  public Instrument(String material, String color, String type, int buyPrice, int sellPrice){
    this.material = material;
    this.color = color;
    this.type = type;
    this.buyPrice = buyPrice;
    this.sellPrice = sellPrice;
  }

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

  public String getColor(){
    return this.color;
  }

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

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

  public int getSellPrice(){
    return this.sellPrice;
  }
}
```

```java
package instrument_management;
import behaviours.*;

public class Guitar extends Instrument implements Playable, Sellable {

  int numberOfStrings;

  public Guitar(String material, String color, String type, int buyPrice, int sellPrice, int
    numberOfStrings){
    super(material, color, type, buyPrice, sellPrice);
    this.numberOfStrings = numberOfStrings;
  }

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

  public String play(){
    return "Aaaa";
  }

  public int calculateMarkup(){
    return sellPrice - buyPrice;
  }

}
```

```java
package behaviours;

public interface Playable {
    String play();
}
```

```java
package instrument_management;
import behaviours.*;
import java.util.*;

public class Shop {

  InstrumentType type;
  String name;
  ArrayList<Playable> instrument;
  ArrayList<Sellable> stock;


  public Shop(InstrumentType type, String name){
    this.type = type;
    this.name = name;
    this.instrument = new ArrayList<Playable>();
    this.stock = new ArrayList<Sellable>();

  }

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

  public void canPlayInTheShop(Playable playable){
    this.instrument.add(playable);
  }

  public int stockCount(){
    return this.stock.size();
  }

  public void stock(Sellable sellable){
    this.stock.add(sellable);
  }

  public void removeOneFromStock(Sellable sellable){
    this.stock.remove(sellable);
  }
```