Evidence for Implementation and Testing Unit.

Ruairidh Grass

E-21

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

```
public abstract class Room {

    private int capacity;
    private ArrayList<Guest> guests;

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

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

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

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

    public void addGuest(Guest guest) {
        if (this.capacity > this.countGuests()){
        this.guests.add(guest);
        }
    }

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

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

An abstract class called “Item”:

```
public abstract class Item implements ISell{

    private String name;
    private int buyPrice;
    private int sellPrice;

    public Item (String name, int buyPrice, int sellPrice) {
        this.name = name;
        this.buyPrice = buyPrice;
        this.sellPrice = sellPrice;
    }
```

Another abstract class called “instrument” which inherits the name, buyPrice and sellPrice from the parent class of “Item”:

```
public abstract class Instrument extends Item implements IPlay{

    private String type;
    private String material;
    private String colour;
    private String sound;

    public Instrument (String name, int buyPrice, int sellPrice, String type, String material, String colour, String sound) {
        super(name, buyPrice, sellPrice);
        this.type = type;
        this.material = material;
        this.colour = colour;
        this.sound = sound;
    }
```

Finally a class called “Guitar which inherits all the parameters from it’s parent class “instrument” in the super constructor:

```
public class Guitar extends Instrument {

    private int stringNumber;

    public Guitar (String name, int buyPrice, int sellPrice, String type, String material, String colour, String sound, int stringNumber) {
        super(name, buyPrice, sellPrice, type, material, colour, sound);
        this.stringNumber = stringNumber;
    }
```

An object in the inherited class:

```
public class GuitarTest {

    Guitar guitar;

    @Before
    public void before() {
        guitar = new Guitar( name: "Acoustic Guitar",  buyPrice: 50,  sellPrice: 100,  type: "String Instrument",  material: "Wood",  colour: "Wood",
  sound: "Pluck Pluck",  stringNumber: 6);
    }
```

Name Method in Parent “Item” Class

```
public abstract class Item implements ISell{

    private String name;
    private int buyPrice;
    private int sellPrice;

    public Item (String name, int buyPrice, int sellPrice) {
        this.name = name;
        this.buyPrice = buyPrice;
        this.sellPrice = sellPrice;
    }

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

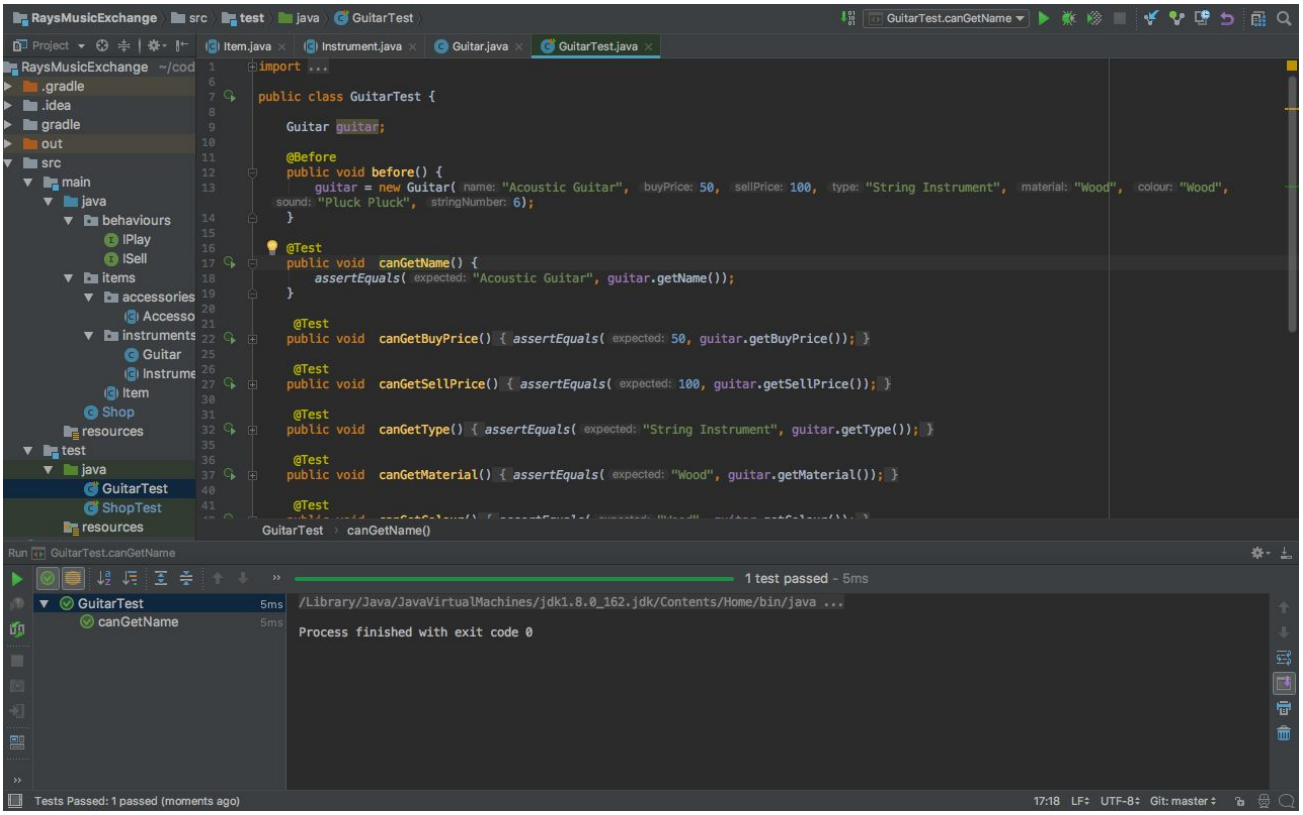
Testing Name Method in “Guitar” class inherited from “Item” class

```
public class GuitarTest {

    Guitar guitar;

    @Before
    public void before() {
        guitar = new Guitar( name: "Acoustic Guitar",  buyPrice: 50,  sellPrice: 100,  type: "String Instrument",  material: "Wood",  colour: "Wood",
  sound: "Pluck Pluck",  stringNumber: 6);
    }

    @Test
    public void  canGetName() { assertEquals( expected: "Acoustic Guitar", guitar.getName()); }
```



I.T 3 - Example of searching

(if you do not have a search and sort algorithm, write one up, take a screenshot. Remember to include the results as well.)

```
def self.all()
  sql = "SELECT * FROM players"
  player_data = SqlRunner.run(sql)
  return player_data.map { |hash| Player.new(hash) }
end
```

```
[2] pry(main)> Player.all
=> [#<Player:0x007ff859c79148 @ability=5, @id=1, @name="Ruri", @strength=2>,
 #<Player:0x007ff859c79030 @ability=1, @id=2, @name="Joe", @strength=4>,
 #<Player:0x007ff859c78f18 @ability=5, @id=3, @name="Ruri", @strength=2>,
 #<Player:0x007ff859c78e00 @ability=1, @id=4, @name="Joe", @strength=4>,
```

I.T 4 – Example of sorting

```ruby
def self.sort_by_wins
  player_wins = self.all
  sorted_players = player_wins.sort { |a, b| b.wins <=> a.wins
  }
  return sorted_players
end
```

```
[1] pry(main)> Player.sort_by_wins
=> [#<Player:0x007ff859b63240 @ability=5, @id=1, @name="Ruri", @strength=2>,
 #<Player:0x007ff859b63060 @ability=5, @id=3, @name="Ruri", @strength=2>,
 #<Player:0x007ff859b62d18 @ability=5, @id=5, @name="Ruri", @strength=2>,
 #<Player:0x007ff859b63150 @ability=1, @id=2, @name="Joe", @strength=4>,
 #<Player:0x007ff859b62f70 @ability=1, @id=4, @name="Joe", @strength=4>,
 #<Player:0x007ff859b62c00 @ability=1, @id=6, @name="Joe", @strength=4>]
```

Highscores

Name	Wins
Ruri	2
Ruri	2
Ruri	2
Joe	0
Joe	0
Joe	0

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

```ruby
fruits = ["apple", "banana", "grape", "orange"]

fruits[1] = "mango" #replaces "banana" with "mango"
#fruits[100] = "peach" #this creates 99 new arrays to place "peach" in number 100
fruits.push("pear") #puts pear on the end
fruits.pop() #removes the last one
fruits.unshift("apple") #copies apple into the start
fruits.shift() #removes first ting in array
# fruits.insert(1, "lemon","lime") #adds new strings to array from 1
```

```ruby
class Arrays < MiniTest::Test

  def test_mango_replaces_banana
    fruits = ["apple", "banana", "grape", "orange"]
    fruits[1] = "mango"
    assert_equal( "mango", fruits[1] )
  end

end
```

```
→  specs ruby arrays_spec.rb
["apple", "mango", "grape", "orange"]
Run options: --seed 43800

# Running:

.

Finished in 0.000955s, 1047.1204 runs/s, 1047.1204 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

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

```ruby
united_kingdom = [
  {
    name: "Scotland",
    population: 5295000,
    capital: "Edinburgh"
  },
  {
    name: "Wales",
    population: 3063000,
    capital: "Swansea"
  },
  {
    name: "England",
    population: 53010000,
    capital: "London"
  }
]
```

```ruby
# Change the capital of Wales from "Swansea" to "Cardiff".
p united_kingdom[1][:capital].replace("Cardiff") #or you can use = "Cardiff"
# Create a Hash for Northern Ireland and add it to the united_kingdom
# array (The capital is Belfast, and the population is 1,811,000).
p united_kingdom << {name:"Northern Ireland", capital:"Belfast", population:1811000}
#could have used united_kingdom.push("northern_ireland")
# Use a loop to print the names of all the countries in the UK.
for country in united_kingdom
  p country[:name]
end
# Use a loop to find the total population of the UK.
total=0
for number in united_kingdom
  total += number[:population]
end
p "The total population is #{total}"
```

```
→  homework git:(master) ✗ ruby exerciseC_day3.rb
"Cardiff"
[{:name=>"Scotland", :population=>5295000, :capital=>"Edinburgh"}, {:name=>"Wales", :population=>3063000, :capital=>"Cardiff"}, {:name=>"England", :population=>53010000, :capital=>"London"}, {:name=>"Northern Ireland", :capital=>"Belfast", :population=>1811000}]
"Scotland"
"Wales"
"England"
"Northern Ireland"
"The total population is 63179000"
```

I.T 7 - Example of polymorphism in a program

“Guitar” and “Ukulele” classes inherit from “Instrument” Class

```
public class Guitar extends Instrument {

    private int stringNumber;

    public Guitar (String name, int buyPrice, int sellPrice, String type, String material, String colour, String sound, int stringNumber) {
        super(name, buyPrice, sellPrice, type, material, colour, sound);
        this.stringNumber = stringNumber;
    }
```

```
public class Ukulele extends Instrument {

    private int stringNumber;

    public Ukulele (String name, int buyPrice, int sellPrice, String type, String material, String colour, String sound, int stringNumber) {
        super(name, buyPrice, sellPrice, type, material, colour, sound);
        this.stringNumber = stringNumber;
    }
```

“Instrument” superclass implements the interface IPlay

```
public abstract class Instrument extends Item implements IPlay{

    private String type;
    private String material;
    private String colour;
    private String sound;

    public Instrument (String name, int buyPrice, int sellPrice, String type, String material, String colour, String sound) {
        super(name, buyPrice, sellPrice);
        this.type = type;
        this.material = material;
        this.colour = colour;
        this.sound = sound;
    }
```

In the “Item” superclass which is the “Instrument’s” parent class you can see that it implements the interface ISell

```
public abstract class Item implements ISell{

    private String name;
    private int buyPrice;
    private int sellPrice;

    public Item (String name, int buyPrice, int sellPrice) {
        this.name = name;
        this.buyPrice = buyPrice;
        this.sellPrice = sellPrice;
    }
```

The Guitar and the Ukulele are Polymorphic as they can be considered as Instrument objects, Item objects, Isell objects or even Iplay objects.