

PDA: Evidence for Implementation and Testing Unit

Verity Ashforth

E19

IT1 - Demonstrate one example of encapsulation you have written in a program

Below is a class Instrument. All of the properties of Instrument: material, colour and instrumentType, are private. This means they cannot be retrieved without the use of a getter method - see the getMaterial(), getColour() and getInstrumentType() methods.

```
package Items.Instruments;

import ...

public class Instrument extends Item implements IPlay {

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

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

    public String play() {
        return null;
    }

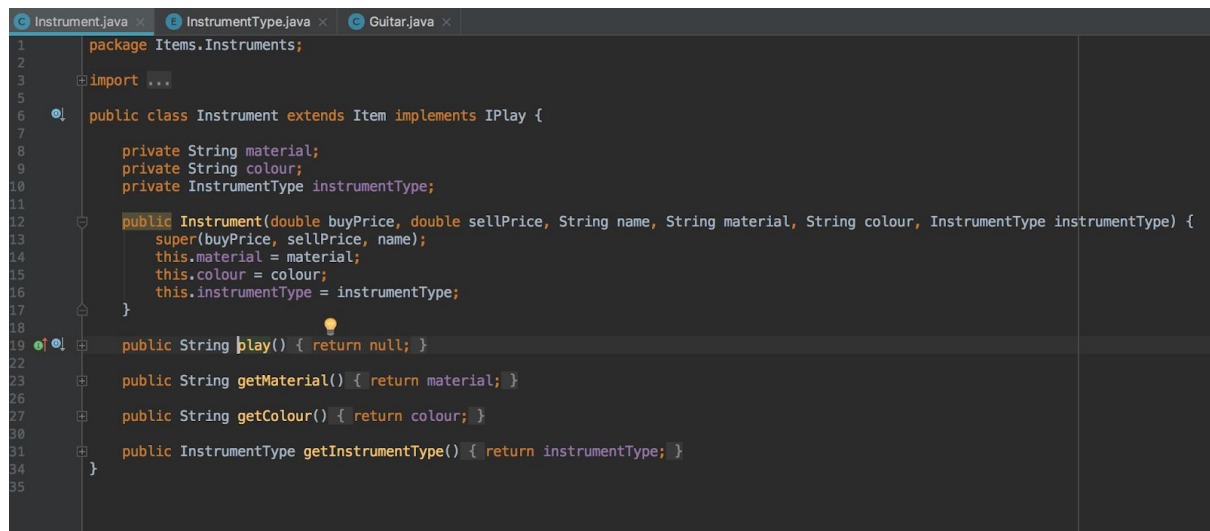
    public String getMaterial() {
        return material;
    }

    public String getColour() {
        return colour;
    }

    public InstrumentType getInstrumentType() {
        return instrumentType;
    }
}
```

IT2 - Example the use of inheritance in a program

The following is an abstract class instrument which is part of a model of a music shop:

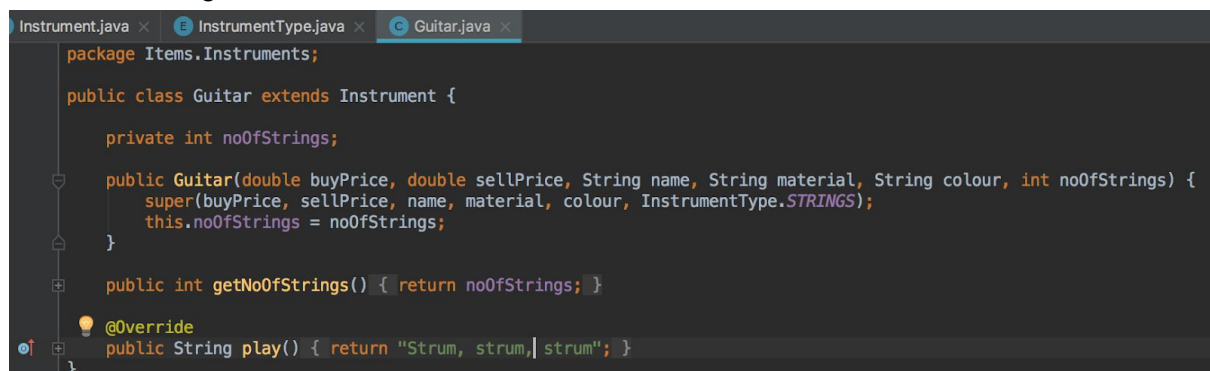


```

1 package Items.Instruments;
2
3 import ...
4
5
6 public class Instrument extends Item implements IPlay {
7
8     private String material;
9     private String colour;
10    private InstrumentType instrumentType;
11
12    public Instrument(double buyPrice, double sellPrice, String name, String material, String colour, InstrumentType instrumentType) {
13        super(buyPrice, sellPrice, name);
14        this.material = material;
15        this.colour = colour;
16        this.instrumentType = instrumentType;
17    }
18
19    public String play() { return null; }
20
21
22    public String getMaterial() { return material; }
23
24
25    public String getColour() { return colour; }
26
27
28    public InstrumentType getInstrumentType() { return instrumentType; }
29
30
31 }
32
33
34
35

```

The shop has many instruments inheriting from the instrument class. Below is an example, a guitar class, which inherits the properties that all instruments have, i.e. the material, the colour and the instrument type. The guitar also has its own property unique to guitars, number of strings.



```

Instrument.java x InstrumentType.java x Guitar.java x
package Items.Instruments;

public class Guitar extends Instrument {

    private int noOfStrings;

    public Guitar(double buyPrice, double sellPrice, String name, String material, String colour, int noOfStrings) {
        super(buyPrice, sellPrice, name, material, colour, InstrumentType.STRING);
        this.noOfStrings = noOfStrings;
    }

    public int getNoOfStrings() { return noOfStrings; }

    @Override
    public String play() { return "Strum, strum,| strum"; }
}

```

The following screenshot shows a number of tests done for the Guitar class using a guitar object :

```
Instrument.java x InstrumentType.java x Guitar.java x GuitarTest.java x
import ...

public class GuitarTest {

    Guitar guitar;

    @Before
    public void before() {
        guitar = new Guitar( buyPrice: 375, sellPrice: 500, name: "Fender", material: "mahogany", colour: "mahogany", noOfStrings: 6);
    }

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

    @Test
    public void canGetBuyPrice() {
        assertEquals( expected: 375, guitar.getBuyPrice(), delta: 0.01);
    }

    @Test
    public void canGetSellPrice() {
        assertEquals( expected: 500, guitar.getSellPrice(), delta: 0.01);
    }

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

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

    @Test
    public void canGetNumberOfStrings() {
        assertEquals( expected: 6, guitar.getNoOfStrings());
    }

    @Test
    public void canGetMarkUp() {
        assertEquals( expected: 125, guitar.calculateMarkUp(), delta: 0.01);
    }
}
```

Below is one of the methods tested above, which uses information from the inherited class:

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

IT3 - Example of Searching

To the left is a screenshot of a table created for the database 'my-money' using PostgreSQL (PSQL).

Directly below is a screenshot of the transactions table sitting in PSQL, viewed in the terminal.

Below is a screenshot of function written in Ruby which will search the transaction table and pull back all transactions of a specified tag (or transaction type) ID. Ruby is able to communicate with PSQL using Sinatra.

```
def Transaction.show_by_type(id)
  sql = "SELECT * FROM transactions
  WHERE tag_id = $1
  ORDER BY trans_date DESC;"
  transactions = SqlRunner.run(sql, [id])
  return transactions.map { |transaction| Transaction.new(transaction) }
end
```

```
[2] pry(main)> Transaction.show_by_type(395)
=> [#<Transaction:0x007fadd9429e18 @amount=60.2, @id=817, @merchant_id=323, @tag_id=395, @trans_date="2018-03-19">,
#<Transaction:0x007fadd9429a58 @amount=60.2, @id=822, @merchant_id=323, @tag_id=395, @trans_date="2018-03-19">,
#<Transaction:0x007fadd9429918 @amount=15.45, @id=824, @merchant_id=325, @tag_id=395, @trans_date="2018-01-15">,
#<Transaction:0x007fadd94297b0 @amount=15.45, @id=829, @merchant_id=325, @tag_id=395, @trans_date="2018-01-15">]
[3] pry(main)> █
```

Running the function `Transaction.show_by_type()` in the terminal using PRY we can see above that the function returns all transactions with `tag_id = 395` as expected.

IT4 - Example of Sorting

```
def Transaction.show_all()
  sql = "SELECT * FROM transactions
  ORDER BY trans_date DESC;"
  transactions = SqlRunner.run(sql)
  return transactions.map { |transaction| Transaction.new(transaction) }
end
```

The function above, written in Ruby, extracts all transactions from the transaction table in PSQL and returns them as instances of the defined class `Transaction` in an array like fashion. As shown below we can assign this a variable name 'transactions'.

```
[3] pry(main)> transactions = Transaction.show_all()
=> [#<Transaction:0x007fadd903f2d0 @amount=12.79, @id=823, @merchant_id=322, @tag_id=398, @trans_date="2018-11-01">,
#<Transaction:0x007fadd903f168 @amount=12.79, @id=818, @merchant_id=322, @tag_id=393, @trans_date="2018-11-01">,
#<Transaction:0x007fadd903f028 @amount=12.79, @id=828, @merchant_id=322, @tag_id=399, @trans_date="2018-11-01">,
#<Transaction:0x007fadd903ee8 @amount=60.2, @id=822, @merchant_id=323, @tag_id=395, @trans_date="2018-03-19">,
#<Transaction:0x007fadd903ed58 @amount=60.2, @id=827, @merchant_id=323, @tag_id=399, @trans_date="2018-03-19">,
#<Transaction:0x007fadd903ebc8 @amount=60.2, @id=817, @merchant_id=323, @tag_id=395, @trans_date="2018-03-19">,
#<Transaction:0x007fadd903eab0 @amount=7.95, @id=816, @merchant_id=325, @tag_id=393, @trans_date="2018-02-09">,
#<Transaction:0x007fadd903e998 @amount=7.95, @id=821, @merchant_id=325, @tag_id=393, @trans_date="2018-02-09">,
#<Transaction:0x007fadd903e880 @amount=7.95, @id=826, @merchant_id=325, @tag_id=393, @trans_date="2018-02-09">,
#<Transaction:0x007fadd903e768 @amount=34.5, @id=815, @merchant_id=324, @tag_id=396, @trans_date="2018-01-30">,
#<Transaction:0x007fadd903e650 @amount=34.5, @id=820, @merchant_id=324, @tag_id=398, @trans_date="2018-01-30">,
#<Transaction:0x007fadd903e538 @amount=34.5, @id=825, @merchant_id=324, @tag_id=394, @trans_date="2018-01-30">,
#<Transaction:0x007fadd903e3d0 @amount=15.45, @id=824, @merchant_id=325, @tag_id=395, @trans_date="2018-01-15">,
#<Transaction:0x007fadd903e2b8 @amount=15.45, @id=819, @merchant_id=325, @tag_id=397, @trans_date="2018-01-15">,
#<Transaction:0x007fadd903e178 @amount=15.45, @id=829, @merchant_id=325, @tag_id=395, @trans_date="2018-01-15">]
```

In Ruby I have written the following function which will sort these transactions by amount either ascending or descending:


```
def Transaction.order_by_amount(transactions, direction)
  transactions_asc = transactions.sort_by {|transaction| transaction.amount }
  transactions_desc = transactions_asc.reverse
  return transactions_asc if direction == "asc"
  return transactions_desc
end
```

Below is what is the result of this function, which sorts the array-like entity transactions as expected:

```
[4] pry(main)> Transaction.order_by_amount(transactions, 'asc')
=> [#<Transaction:0x007fadd903e880 @amount=7.95, @id=826, @merchant_id=325, @tag_id=393, @trans_date="2018-02-09">,
#<Transaction:0x007fadd903e998 @amount=7.95, @id=821, @merchant_id=325, @tag_id=393, @trans_date="2018-02-09">,
#<Transaction:0x007fadd903eab0 @amount=7.95, @id=816, @merchant_id=325, @tag_id=393, @trans_date="2018-02-09">,
#<Transaction:0x007fadd903f2d0 @amount=12.79, @id=823, @merchant_id=322, @tag_id=398, @trans_date="2018-11-01">,
#<Transaction:0x007fadd903f168 @amount=12.79, @id=818, @merchant_id=322, @tag_id=393, @trans_date="2018-11-01">,
#<Transaction:0x007fadd903f028 @amount=12.79, @id=828, @merchant_id=322, @tag_id=399, @trans_date="2018-11-01">,
#<Transaction:0x007fadd903e178 @amount=15.45, @id=829, @merchant_id=325, @tag_id=395, @trans_date="2018-01-15">,
#<Transaction:0x007fadd903e3d0 @amount=15.45, @id=824, @merchant_id=325, @tag_id=395, @trans_date="2018-01-15">,
#<Transaction:0x007fadd903e2b8 @amount=15.45, @id=819, @merchant_id=325, @tag_id=397, @trans_date="2018-01-15">,
#<Transaction:0x007fadd903e650 @amount=34.5, @id=820, @merchant_id=324, @tag_id=398, @trans_date="2018-01-30">,
#<Transaction:0x007fadd903e768 @amount=34.5, @id=815, @merchant_id=324, @tag_id=396, @trans_date="2018-01-30">,
#<Transaction:0x007fadd903e538 @amount=34.5, @id=825, @merchant_id=324, @tag_id=394, @trans_date="2018-01-30">,
#<Transaction:0x007fadd903ebc8 @amount=60.2, @id=817, @merchant_id=323, @tag_id=395, @trans_date="2018-03-19">,
#<Transaction:0x007fadd903eee8 @amount=60.2, @id=822, @merchant_id=323, @tag_id=395, @trans_date="2018-03-19">,
#<Transaction:0x007fadd903ed58 @amount=60.2, @id=827, @merchant_id=323, @tag_id=399, @trans_date="2018-03-19">]
[5] pry(main)> Transaction.order_by_amount(transactions, 'desc')
=> [#<Transaction:0x007fadd903ed58 @amount=60.2, @id=827, @merchant_id=323, @tag_id=399, @trans_date="2018-03-19">,
#<Transaction:0x007fadd903eee8 @amount=60.2, @id=822, @merchant_id=323, @tag_id=395, @trans_date="2018-03-19">,
#<Transaction:0x007fadd903ebc8 @amount=60.2, @id=817, @merchant_id=323, @tag_id=395, @trans_date="2018-03-19">,
#<Transaction:0x007fadd903e538 @amount=34.5, @id=825, @merchant_id=324, @tag_id=394, @trans_date="2018-01-30">,
#<Transaction:0x007fadd903e768 @amount=34.5, @id=815, @merchant_id=324, @tag_id=396, @trans_date="2018-01-30">,
#<Transaction:0x007fadd903e650 @amount=34.5, @id=820, @merchant_id=324, @tag_id=398, @trans_date="2018-01-30">,
#<Transaction:0x007fadd903e2b8 @amount=15.45, @id=819, @merchant_id=325, @tag_id=397, @trans_date="2018-01-15">,
#<Transaction:0x007fadd903e3d0 @amount=15.45, @id=824, @merchant_id=325, @tag_id=395, @trans_date="2018-01-15">,
#<Transaction:0x007fadd903e178 @amount=15.45, @id=829, @merchant_id=325, @tag_id=395, @trans_date="2018-01-15">,
#<Transaction:0x007fadd903f2d0 @amount=12.79, @id=823, @merchant_id=322, @tag_id=399, @trans_date="2018-11-01">,
#<Transaction:0x007fadd903f168 @amount=12.79, @id=818, @merchant_id=322, @tag_id=393, @trans_date="2018-11-01">,
#<Transaction:0x007fadd903f2d0 @amount=12.79, @id=823, @merchant_id=322, @tag_id=398, @trans_date="2018-11-01">,
#<Transaction:0x007fadd903eab0 @amount=7.95, @id=816, @merchant_id=325, @tag_id=393, @trans_date="2018-02-09">,
#<Transaction:0x007fadd903e998 @amount=7.95, @id=821, @merchant_id=325, @tag_id=393, @trans_date="2018-02-09">,
#<Transaction:0x007fadd903e880 @amount=7.95, @id=826, @merchant_id=325, @tag_id=393, @trans_date="2018-02-09">]
```

IT5 - An example of an array, a function that uses an array, and the result.

Below is an example of an array which has been assigned the variable name 'vegetables' and a method which will sort an array of strings alphabetically and print the sorted array.

```
1  vegetables = ["celery", "carrot", "turnip", "pepper", "onion"]
2
3  def order_strings_array_alphabetically(array)
4    sorted_array = array.sort
5    p sorted_array
6  end
7
8
9  p vegetables
0  order_strings_array_alphabetically(vegetables)
```

Below is the result outputted to the terminal both of printing the array vegetables and underneath the result of the sorting method.

```
➔ PDA Work ruby IT_examples.rb
["celery", "carrot", "turnip", "pepper", "onion"]
["carrot", "celery", "onion", "pepper", "turnip"]
```

IT6 - An example of a hash, a function that uses a hash, and the result.

Below is an example of a hash of staff names along with their staff numbers. And a method which will print a statement connecting each of the keys with their pairs.

```
8 staff_nos = {
9   David: 12345,
10  Hayley: 12346,
11  Simone: 12346,
12  Colette: 12347,
13  Alistair: 12348
14 }
15
16 def pair_keys_and_values(hash)
17   hash.each do |key, value|
18     p "#{key} is connected to #{value}"
19   end
20 end
21
22 pair_keys_and_values(staff_nos)
```

The screenshot below is the result to the terminal:

```
➔ PDA Work ruby IT_examples.rb
"David is connected to 12345"
"Hayley is connected to 12346"
"Simone is connected to 12346"
"Colette is connected to 12347"
"Alistair is connected to 12348"
```

IT7 - Example of Polymorphism in a program.

```
Apple.java x Grocery.java x Toothpaste.java x Customer.java x Cereal.java x
1 package Groceries;
2
3 public class Toothpaste extends Grocery{
4     private String name;
5     private double price;
6     private String barcode;
7     private String whiteningLevel;
8
9
10    public Toothpaste(String name, double price, String barcode, String whiteningLevel) {
11        super(name, price, barcode);
12        this.whiteningLevel = whiteningLevel;
13    }
14
15
16 }
```

```
Apple.java x Grocery.java x Toothpaste.java x Customer.java x Cereal.java x
1 package Groceries;
2
3 public class Cereal extends Grocery{
4
5     private String name;
6     private double price;
7     private String barcode;
8     private String boxSize;
9
10
11    public Cereal(String name, double price, String barcode, String boxSize) {
12        super(name, price, barcode);
13        this.boxSize = boxSize;
14    }
15
16
17 }
```

```
Apple.java x Grocery.java x Toothpaste.java x Customer.java x Cereal.java x
1 package Groceries;
2
3 public class Apple extends Grocery {
4
5     private String name;
6     private double price;
7     private String barcode;
8     private double weight;
9
10
11    public Apple(String name, double price, String barcode, double weight) {
12        super(name, price, barcode);
13        this.weight = weight;
14    }
15
16
17 }
```

In this example we have an abstract superclass Grocery and three types of grocery, apple, cereal and toothpaste, all of which have their own class extending from Grocery. As such

they share the common properties of a Grocery: name, price and barcode. Grocery, and by abstraction apple, cereal and toothpaste, all implement the interface IBuyable.

```
package Groceries;

public abstract class Grocery implements IBuyable {

    private String name;
    private double price;
    private String barcode;

    public Grocery(String name, double price, String barcode) {
        this.name = name;
        this.price = price;
        this.barcode = barcode;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    public String getBarcode() {
        return barcode;
    }
}
```

The customer has a method to purchase a Grocery, which will add the IBuyable grocery to their ArrayList of groceries. Because Apple, Toothpaste and Cereal are all extensions of Grocery which implements IBuyable we can plug in any object instance of either class.

```
Apple.java × | Grocery.java × | IBuyable.java ×
package Groceries;

public interface IBuyable {
    |
}
```

We do not need to write separate methods to purchase apples, toothpaste and cereal. This is an example of polymorphism.


```
Apple.java × Grocery.java × IBuyable.java × Toothpaste.java × Cu
package Customer;

import Groceries.Grocery;
import Groceries.IBuyable;

import java.util.ArrayList;

public class Customer {

    private String name;
    private double wallet;
    private ArrayList<IBuyable> purchasedGroceries;

    public Customer(String name, double wallet) {
        this.name = name;
        this.wallet = wallet;
    }

    public String getName() {
        return name;
    }

    public double getWallet() {
        return wallet;
    }

    public void purchaseGrocery(IBuyable grocery) {
        this.purchasedGroceries.add(grocery);
    }

}
```

