

I.T 1 An Example of Encapsulation (function from a Class)

```
public boolean isValidMove(Card moveCard, Card targetCard) {  
    int moveValue = deck.getCardValue(moveCard);  
    int targetValue = deck.getCardValue(targetCard);  
    String moveColour = deck.getSuitColour(moveCard);  
    String targetColour = deck.getSuitColour(targetCard);  
  
    if(moveCard.isRevealed() && targetCard.isRevealed()){  
        if(targetValue == (moveValue+1)){  
            if(moveColour != targetColour){  
                return true;  
            }  
        }  
    }  
  
    return false;  
}
```

I.T Screenshots for use of Inheritance

i. A Super Class

```

1  package music_management;
2  import behaviours.*;
3
4  ▼ public abstract class Instrument implements Playable, Sellable{
5
6      private InstrumentColour colour;
7      private InstrumentType type;
8      private int wholesalePrice;
9      private int retailPrice;
10
11  ▼ public Instrument(InstrumentColour colour, InstrumentType type,
12      int wholesalePrice, int retailPrice){
13
14      this.colour = colour;
15      this.type = type;
16      this.wholesalePrice = wholesalePrice;
17      this.retailPrice = retailPrice;
18  }
19
20  public int calculateMarkUp(){
21      return this.retailPrice - this.wholesalePrice;
22  }
23
24
25  }

```

ii. Child class (inherits from Instrument)

```

1  package music_management;
2
3  public class FrenchHorn extends Instrument{
4
5      public FrenchHorn(InstrumentType type,
6          InstrumentColour colour, int wholesalePrice,
7          int retailPrice ){
8          super(colour, type, wholesalePrice, retailPrice);
9      }
10
11      public String play(){
12          return "Broot";
13      }
14  }

```

iii. An object of the child class

```
1  import static org.junit.Assert.*;
2  import org.junit.*;
3  import behaviours.*;
4  import music_management.*;
5
6  public class FrenchHornTest{
7
8      FrenchHorn horn;
9
10     @Before
11     public void before(){
12         horn = new FrenchHorn(InstrumentType.WIND, InstrumentColour.GOLD, 300, 500);
13     }
14
15     @Test
16     public void canGetMarkUp(){
17         assertEquals(200, horn.calculateMarkUp());
18     }
19 }
```

iv. A method that uses the information from the inherited class (ArrayList contains Instruments of Sellable types)

```
1 package music_management;
2 import java.util.*;
3 import behaviours.*;
4
5 public class Shop{
6     private ArrayList<Sellable> stock;
7
8     public Shop(){
9         this.stock = new ArrayList<Sellable>();
10    }
11
12    public int totalPotentialProfit(){
13        int potentialProfit = 0;
14        for(Sellable item: this.stock){
15            potentialProfit += item.calculateMarkUp();
16        }
17        return potentialProfit;
18    }
19 }
```

I.T 3 Search function

```
binary_search_ruby.rb *
1 class BinarySearch
2   attr_reader :array
3   def initialize()
4     @array = ['a', 'b', 'c', 'd', 'e']
5
6   end
7
8   def binary_search(item_to_find)
9     high = (@array.size) - 1
10    low = 0
11
12    while (low <= high)
13      mid = (high + low) / 2
14
15      if @array[mid] == item_to_find
16        return @array[mid]
17      elsif @array[mid] < item_to_find
18        low = mid + 1
19      else
20        high = mid - 1
21      end
22    end
23    return nil
24  end
25 end
26
27
28 search = BinarySearch.new
29 p search.binary_search('b')
```

```
example_code — user@CODECLAN059 — -zsh — 80x24
../example_code      psql  +
[→ example_code git:(master) * ruby binary_search_ruby.rb
"b"
→ example_code git:(master) * ]
```

I.T 4 Sort Function

```
own_sort.rb
* 1 require('pry')
* 2 class OwnSearch
* 3
* 4   def self.sorting(array)
* 5     return array.first if array.size <= 1
* 6     sorted_arr = Array.new()
* 7     #iterate through the given array and each iteration find the largest and smallest values
* 8     while array.size > 0
* 9       count = 0
*10       smallest = array[0]
*11       smallest_index = 0
*12       array.each do |item|
*13         smallest_index = count if item < smallest
*14         smallest = item if item < smallest
*15         count += 1
*16       end
*17       sorted_arr.push(array[smallest_index])
*18       array.delete_at(smallest_index)
*19     end
*20     return sorted_arr
*21   end
*22 end
*23
*24 p OwnSearch.sorting(['q','c','z','b', 's', 'd'])
*25 p OwnSearch.sorting([9,8,7,6,5,4,3,2,1])
```

```
pda — user@CODECLAN059 — -zsh — 80x26
..clan_work/pda      psql  +
[→ pda git:(master) × ruby example_code/week_3/own_sort.rb
["b", "c", "d", "q", "s", "z"]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
→ pda git:(master) × █
```

I.T 5 Use of an array in a program

```
my_array.rb — example_code
my_array.rb
1 def count_the_kittens(array_of_cats)
2   return array_of_cats.select{|feline| feline == "kitten"}.count
3 end
4
5 p count_the_kittens(["cat", "cat", "kitten", "kitten", "cat", "cat", "kitten"])
```



```
example_code — user@CODECLAN059 — ../example_code — zsh — 80×24
[→ example_code ruby my_array.rb
3
→ example_code █
```

I.T 6 Use of a hash in a program

```
runner.rb
1 require_relative('person.rb')
2 require_relative('medic.rb')
3 require_relative('agent.rb')
4
5
6 person = Person.new("Chris", "Hunter")
7 medic = Medic.new("Steph", "Beattie")
8 agent = Agent.new("James", "Blonde")
9 hash_of_people = {
10   person1: person,
11   person2: medic,
12   person3: agent
13 }
14
15
16 hash_of_people.each{|num, person | p person.talk}
```



```
own_inheritance — user@CODECLAN059 — ..n_inheritance — -zsh — 80x24
[→ own_inheritance ruby runner.rb
"Hi, I'm Chris Hunter"
"Hi, I'm Steph Beattie"
"The names Blonde, James Blonde"
→ own_inheritance █
```

I.T 7 Use of Polymorphism

```
Sellable.java
1 package behaviours;
2
3 public interface Sellable{
4     int calculateMarkUp();
5 }
```

An Item can is an Item or can be a Sellable.

```

package music_management;
import behaviours.*;

public class Item implements Sellable{

    String item;
    int wholesalePrice;
    int retailPrice;

    public Item(String item, int wholesalePrice, int retailPrice){
        this.item = item;
        this.wholesalePrice = wholesalePrice;
        this.retailPrice = retailPrice;
    }

    public int calculateMarkUp(){
        return this.retailPrice - this.wholesalePrice;
    }

}

```

Instrument super class which implements Sellable.

```

package music_management;
import behaviours.*;

public abstract class Instrument implements Playable, Sellable{

    private InstrumentColour colour;
    private InstrumentType type;
    private int wholesalePrice;
    private int retailPrice;

    public Instrument(InstrumentColour colour, InstrumentType type, int wholesalePrice, int retailPrice){

        this.colour = colour;
        this.type = type;
        this.wholesalePrice = wholesalePrice;
        this.retailPrice = retailPrice;
    }

    public int calculateMarkUp(){
        return this.retailPrice - this.wholesalePrice;
    }

}

```

A FrenchHorn is an Instrument but can be a Sellable.

```
Shop.java      ShopTest.java      FrenchHorn.java
1 package music_management;
2
3 public class FrenchHorn extends Instrument{
4
5     public FrenchHorn(InstrumentType type, InstrumentColour colour, int wholesalePrice, int
       retailPrice ){
6         super(colour, type, wholesalePrice, retailPrice);
7     }
8
9     public String play(){
10         return "Broot";
11     }
12 }
```

Shop class which uses an ArrayList of Sellable types.

```
Shop.java
1 package music_management;
2 import java.util.*;
3 import behaviours.*;
4
5 public class Shop{
6     private ArrayList<Sellable> stock;
7
8     public Shop(){
9         this.stock = new ArrayList<Sellable>();
10     }
11
12     public int totalPotentialProfit(){
13         int potentialProfit = 0;
14         for(Sellable item: this.stock){
15             potentialProfit += item.calculateMarkUp();
16         }
17         return potentialProfit;
18     }
19
20     public int countStock(){
21         return stock.size();
22     }
23
24     public void addStock(Sellable item){
25         stock.add(item);
26     }
27 }
```

FrenchHorn and Item objects can be added to shop object stock as Sellable. Then test that totalPotentialProfit() adds all calculateMarkUp() functions for each Sellable in the ArrayList.

```

52
53 @Test
54 public void canGetTotalPotentialProfitFromItemsAndInstruments(){
55     FrenchHorn horn = new FrenchHorn(InstrumentType.WIND, InstrumentColour.GOLD, 300, 500);
56     FrenchHorn horn2 = new FrenchHorn(InstrumentType.WIND, InstrumentColour.SILVER, 300, 500);
57     Item item = new Item("Sheet Music", 10, 20);
58     shop.addStock(horn);
59     shop.addStock(horn2);
60     shop.addStock(item);
61     assertEquals(410, shop.totalPotentialProfit());
62     // horn markup = 200, horn2 markup = 200 and item markup = 10 total = 410
63
64 }
65 }

```

Tests run as expected.

A terminal window titled "music_shop_homework — user@CODECLAN059 — -zsh — 87x25". The window shows the output of running tests in the "shop_homework" directory. The output indicates that all tests passed successfully.

```

..shop_homework ..3/composition +

OK (1 test)

Running GuitarTest
JUnit version 4.12
..
Time: 0.006

OK (2 tests)

Running ItemTest
JUnit version 4.12
.
Time: 0.004

OK (1 test)

Running ShopTest
JUnit version 4.12
.....
Time: 0.009

OK (6 tests)

→ music_shop_homework git:(master) ×

```


