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

```java
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;
  }


}
```

ii. Child class (inherits from Instrument)

```java
package music_management;

public class FrenchHorn extends Instrument{

  public FrenchHorn(InstrumentType type,
    InstrumentColour colour, int wholesalePrice,
    int retailPrice ){
    super(colour, type, wholesalePrice, retailPrice);
  }

  public String play(){
    return "Broot";
  }
}
```

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 }
```

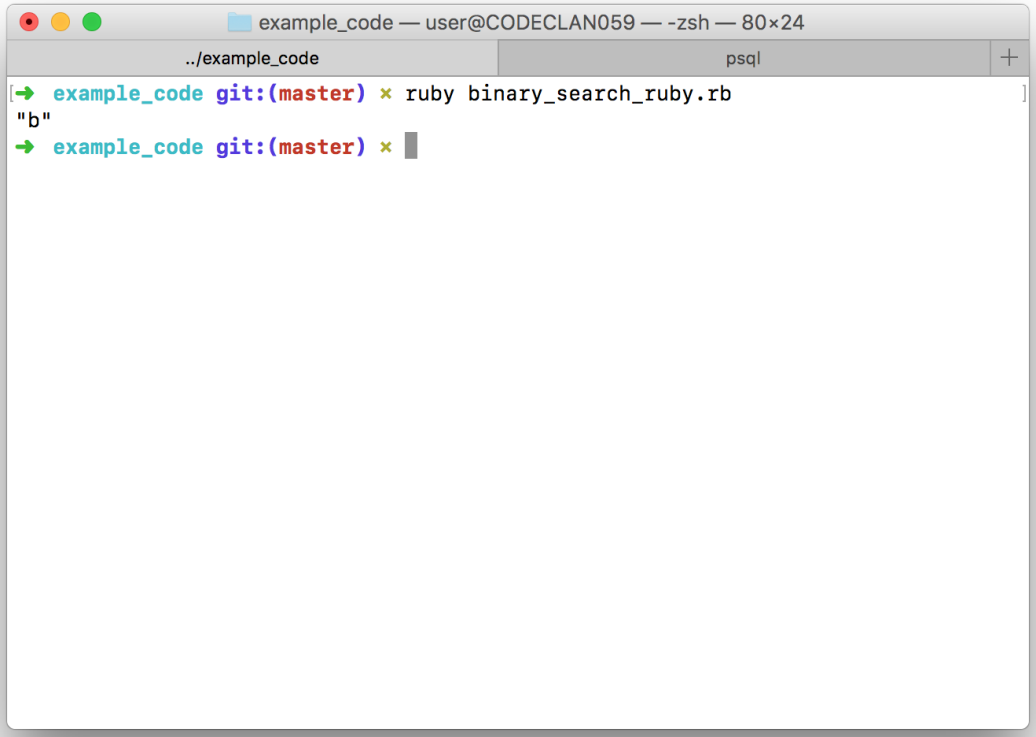
iv. A method that uses the information from the inherited class (ArrayList contains Instruments of Sellable types)

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

public class Shop{
  private ArrayList<Sellable> stock;

  public Shop(){
    this.stock = new ArrayList<Sellable>();
  }

  public int totalPotentialProfit(){
    int potentialProfit = 0;
    for(Sellable item: this.stock){
      potentialProfit += item.calculateMarkUp();
    }
    return potentialProfit;
  }
```

I.T 3 Search function

binary_search_ruby.rb

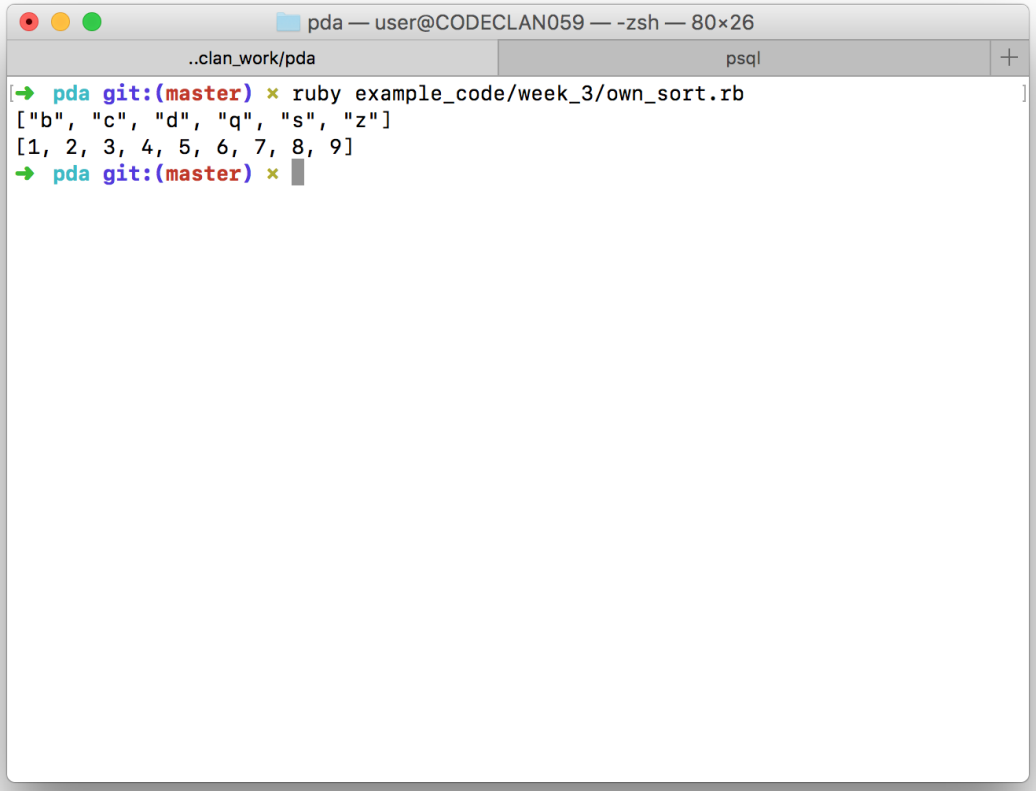
```ruby
class BinarySearch
  attr_reader :array
  def initialize()
    @array = ['a', 'b', 'c' , 'd', 'e']

  end

  def binary_search(item_to_find)
    high = (@array.size) - 1
    low = 0

    while (low <= high)
      mid = (high + low) / 2

      if @array[mid] == item_to_find
        return array[mid]
      elsif @array[mid] < item_to_find
        low = mid + 1
      else
        high = mid - 1
      end
    end
    return nil
  end
end


search = BinarySearch.new
p search.binary_search('b')
```



I.T 4 Sort Function

own_sort.rb

```ruby
require('pry')
class OwnSearch

  def self.sorting(array)
    return array.first if array.size <= 1
    sorted_arr = Array.new()
    #iterate through the given array and each iteration find the largest and smallest values
    while array.size > 0
      count = 0
      smallest = array[0]
      smallest_index = 0
      array.each do |item|
        smallest_index = count if item < smallest
        smallest = item if item < smallest
        count += 1
      end
      sorted_arr.push(array[smallest_index])
      array.delete_at(smallest_index)
    end
    return sorted_arr
  end
end

p OwnSearch.sorting(['q','c','z','b', 's', 'd'])
p OwnSearch.sorting([9,8,7,6,5,4,3,2,1])
```

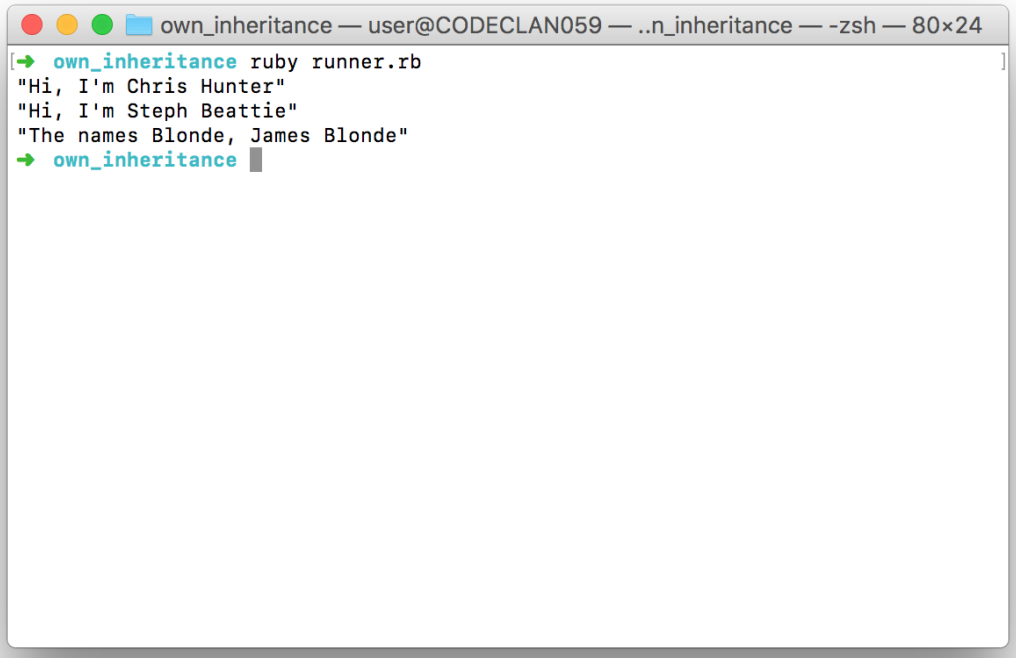


I.T 5 Use of an array in a program

I.T 6 Use of a hash in a program

runner.rb

```ruby
require_relative('person.rb')
require_relative('medic.rb')
require_relative('agent.rb')


person = Person.new("Chris", "Hunter")
medic  = Medic.new("Steph", "Beattie")
agent = Agent.new("James", "Blonde")
hash_of_people = {
  person1: person,
  person2: medic,
  person3: agent

}

hash_of_people.each{|num, person | p person.talk}
```



I.T 7 Use of Polymorphism

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

```java
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.

```java
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

```java
package music_management;

public class FrenchHorn extends Instrument{

  public FrenchHorn(InstrumentType type, InstrumentColour colour, int wholesalePrice, int
    retailPrice ){
    super(colour, type, wholesalePrice, retailPrice);
  }

  public String play(){
    return "Broot";
  }
}
```

Shop class which uses an ArrayList of Sellable types.

Shop.java

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

public class Shop{
  private ArrayList<Sellable> stock;

  public Shop(){
    this.stock = new ArrayList<Sellable>();
  }

  public int totalPotentialProfit(){
    int potentialProfit = 0;
    for(Sellable item: this.stock){
      potentialProfit += item.calculateMarkUp();
    }
    return potentialProfit;
  }

  public int countStock(){
    return stock.size();
  }

  public void addStock(Sellable item){
    stock.add(item);
  }
```

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 cancanGetTotalPotentialProfitFromItemsAndInstruments(){
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.