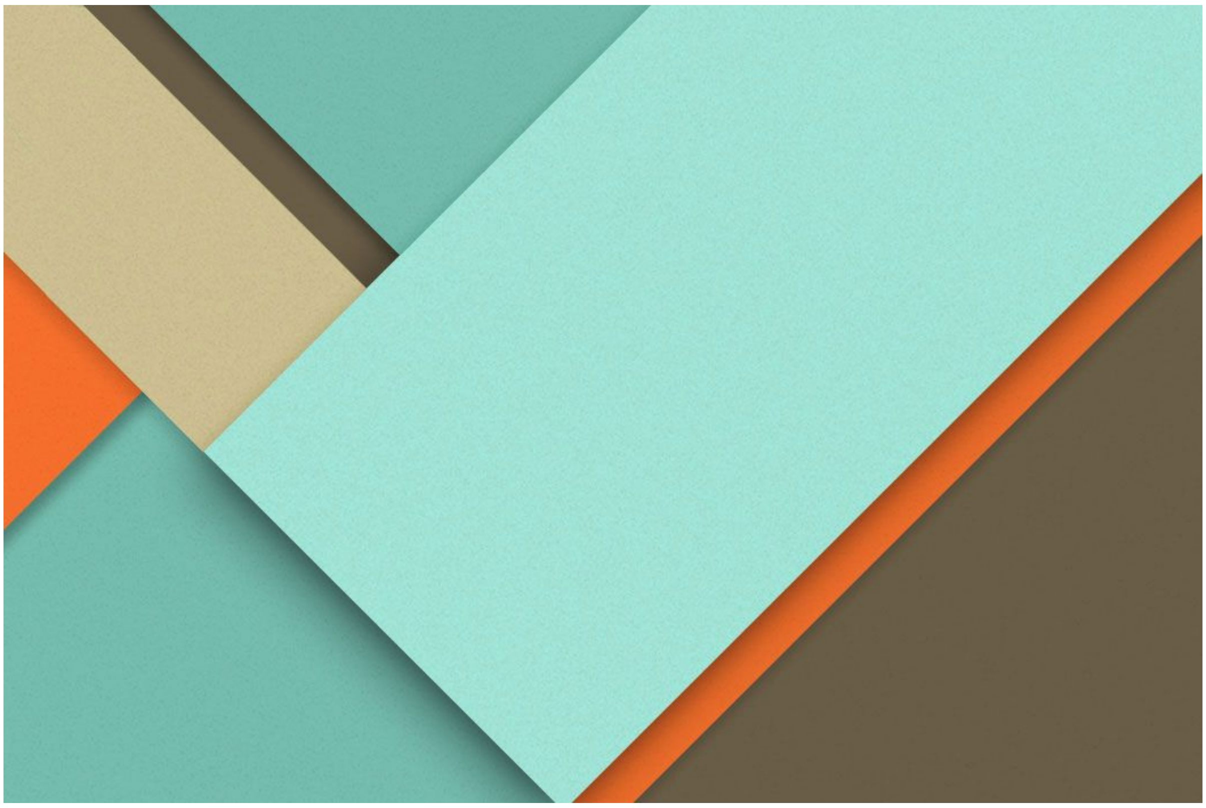


PDA - Implementation and Testing Unit

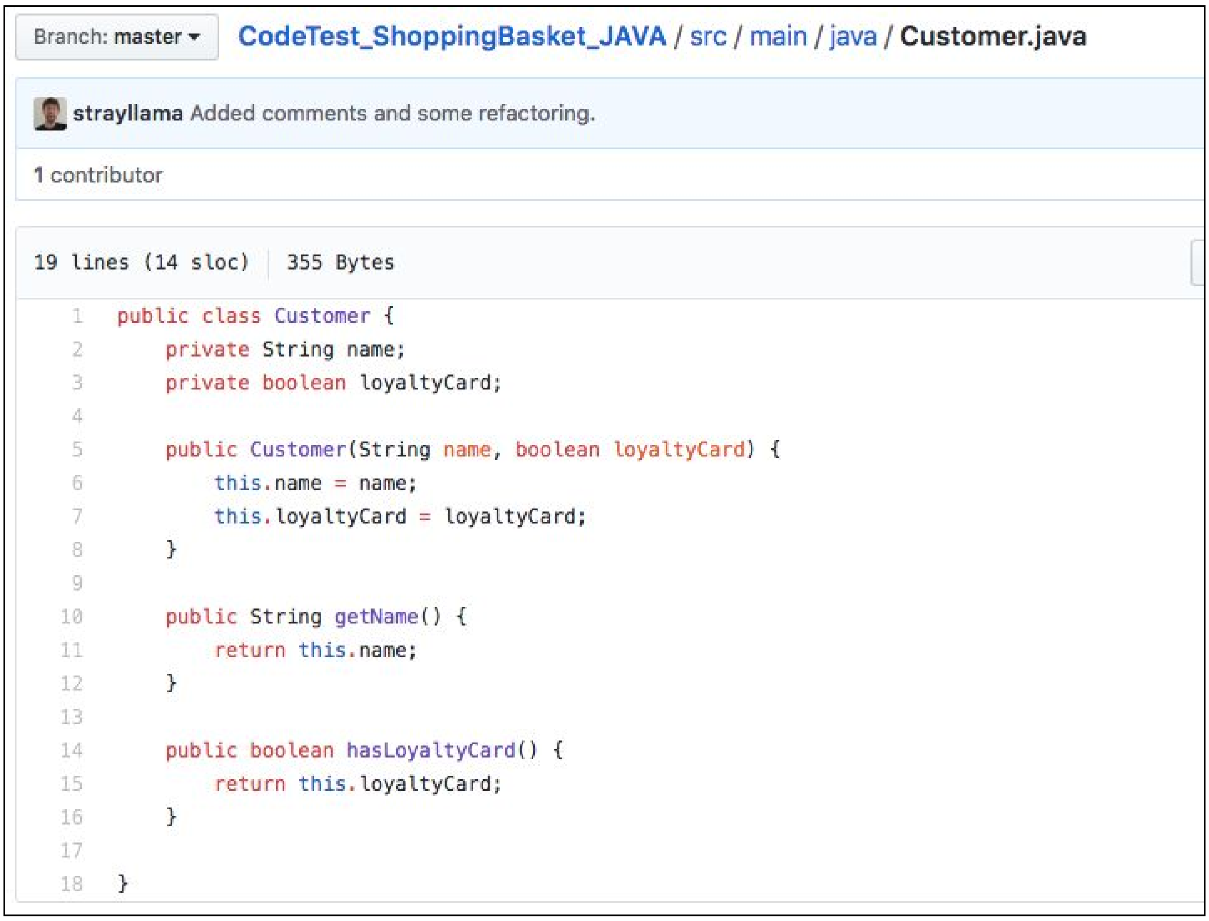
CodeClan Course Evidence

12.03.2018

Paul Clatworthy
2 Liberton Tower Lane, Edinburgh, EH16 6TQ

I.T 1 - Screenshot of encapsulation in a program:

- A class that can not have its state altered from outside and the state can only be seen by calling its methods to return it.



Branch: master ▾ CodeTest_ShoppingBasket_JAVA / src / main / java / Customer.java

strayllama Added comments and some refactoring.

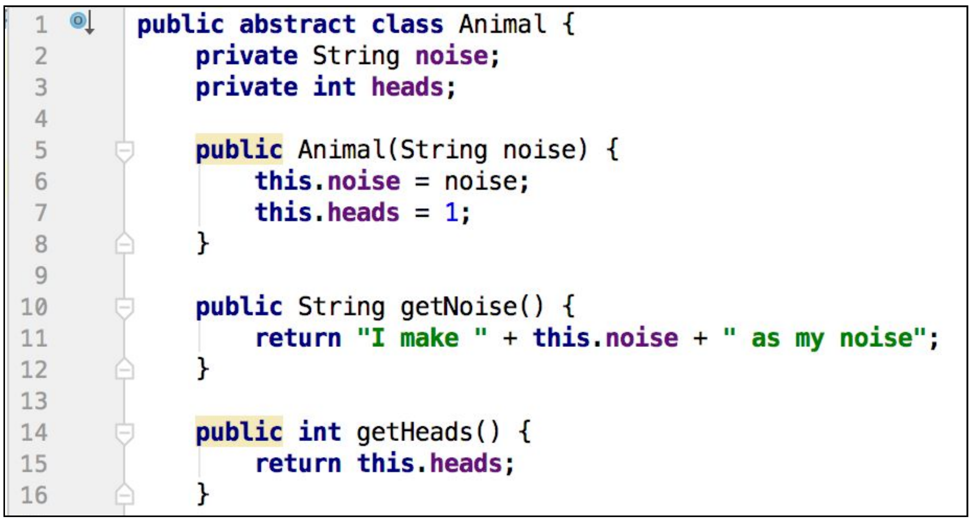
1 contributor

19 lines (14 sloc) | 355 Bytes

```java
public class Customer {
    private String name;
    private boolean loyaltyCard;

    public Customer(String name, boolean loyaltyCard) {
        this.name = name;
        this.loyaltyCard = loyaltyCard;
    }

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

    public boolean hasLoyaltyCard() {
        return this.loyaltyCard;
    }

}
```

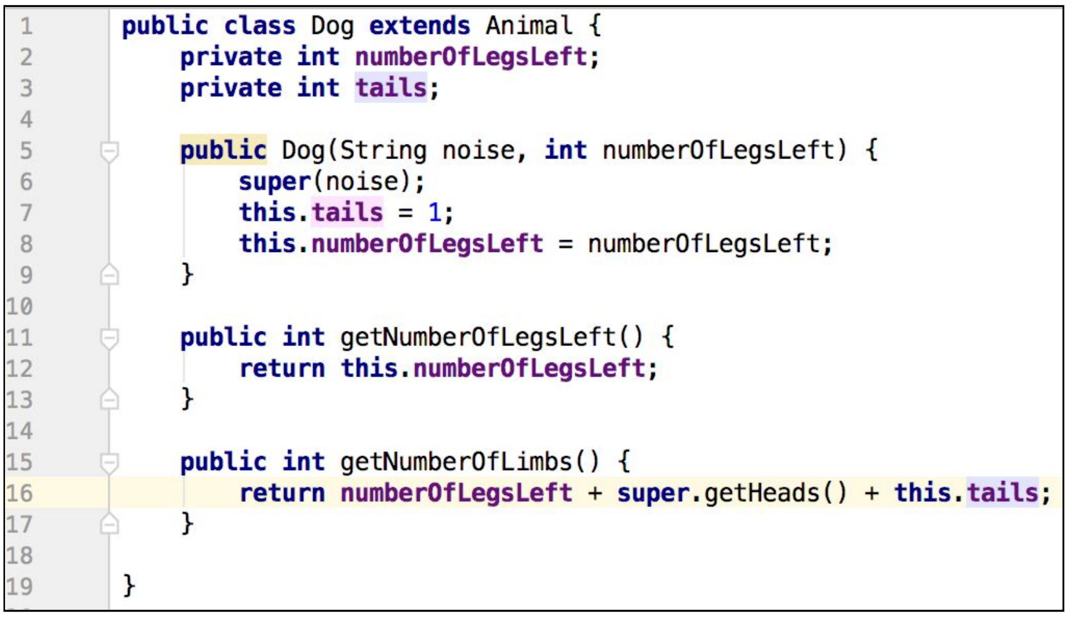
I.T 2 - Screenshot of the use of Inheritance in a program:

- An Abstract Class



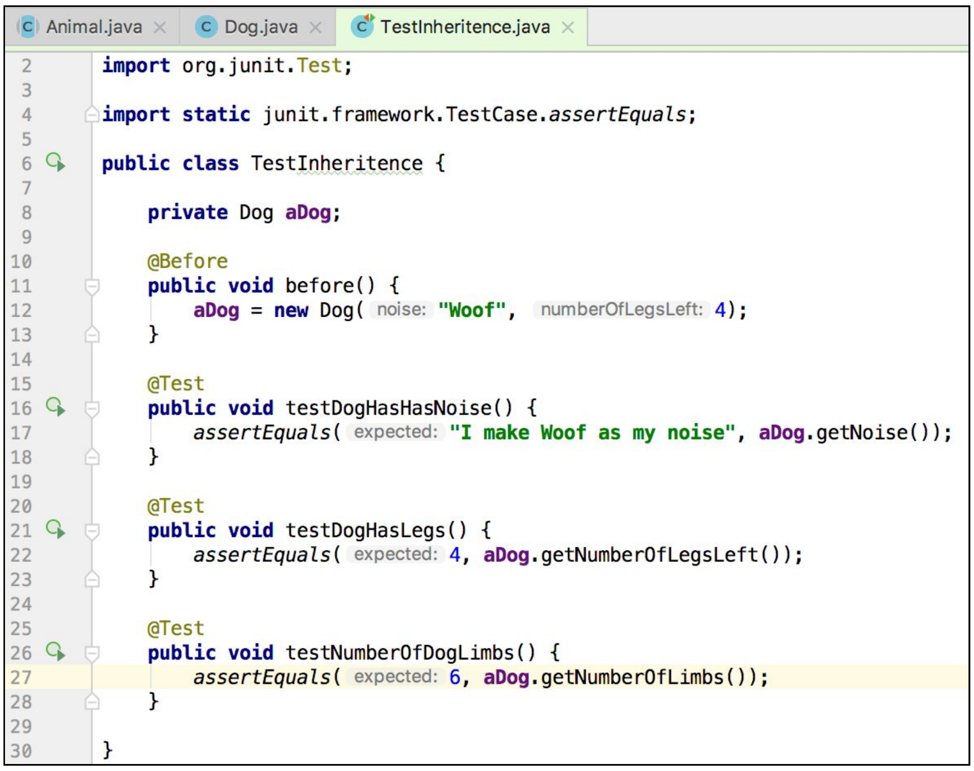
```java
public abstract class Animal {
    private String noise;
    private int heads;

    public Animal(String noise) {
        this.noise = noise;
        this.heads = 1;
    }

    public String getNoise() {
        return "I make " + this.noise + " as my noise";
    }

    public int getHeads() {
        return this.heads;
    }
```

- A Class that inherits from the previous abstract class AND a Method that uses the information inherited from another class:



```java
public class Dog extends Animal {
    private int numberOfLegsLeft;
    private int tails;

    public Dog(String noise, int numberOfLegsLeft) {
        super(noise);
        this.tails = 1;
        this.numberOfLegsLeft = numberOfLegsLeft;
    }

    public int getNumberOfLegsLeft() {
        return this.numberOfLegsLeft;
    }

    public int getNumberOfLimbs() {
        return numberOfLegsLeft + super.getHeads() + this.tails;
    }

}
```

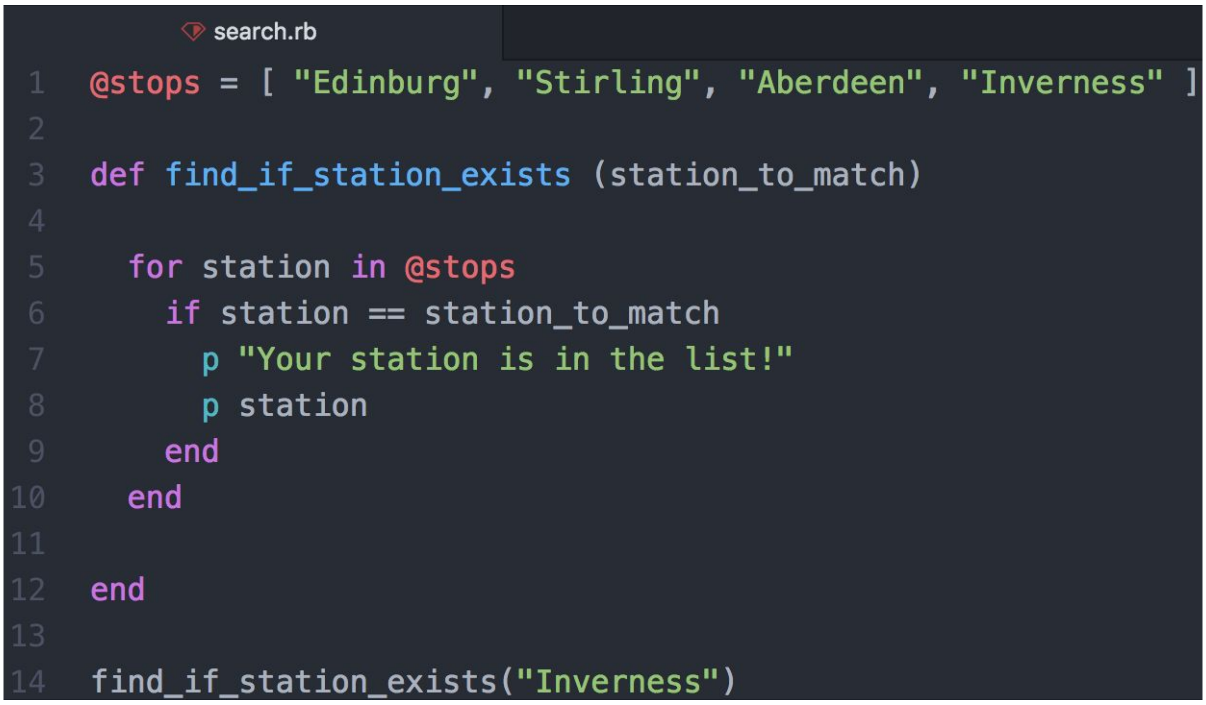
- An Object in the inherited class



```java
import org.junit.Test;

import static junit.framework.TestCase.assertEquals;

public class TestInheritence {

    private Dog aDog;

    @Before
    public void before() {
        aDog = new Dog( noise: "Woof", numberOfLegsLeft: 4);
    }

    @Test
    public void testDogHasHasNoise() {
        assertEquals( expected: "I make Woof as my noise", aDog.getNoise());
    }

    @Test
    public void testDogHasLegs() {
        assertEquals( expected: 4, aDog.getNumberOfLegsLeft());
    }

    @Test
    public void testNumberOfDogLimbs() {
        assertEquals( expected: 6, aDog.getNumberOfLimbs());
    }

}
```

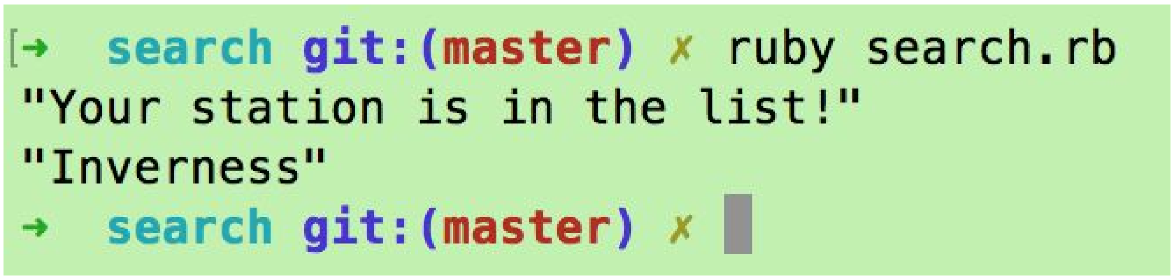
I.T 3 - Demonstrate searching data in a program. Take screenshots of:

- A function that **searches data**:



```ruby
@stops = [ "Edinburg", "Stirling", "Aberdeen", "Inverness" ]

def find_if_station_exists (station_to_match)

  for station in @stops
    if station == station_to_match
      p "Your station is in the list!"
      p station
    end
  end

end

find_if_station_exists("Inverness")
```

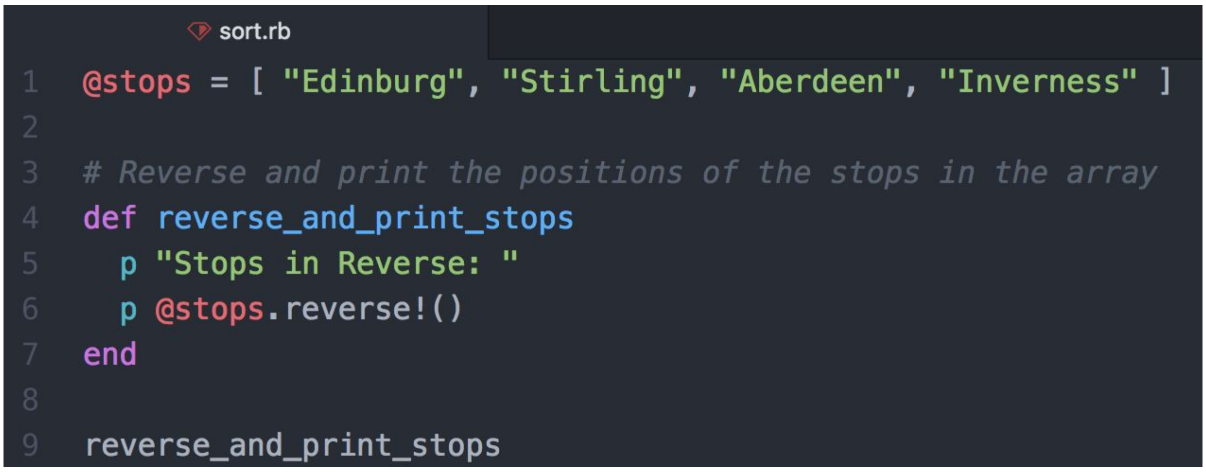
- The result of the function running:



```
[→  search git:(master) ✗ ruby search.rb
"Your station is in the list!"
"Inverness"
→  search git:(master) ✗ 
```

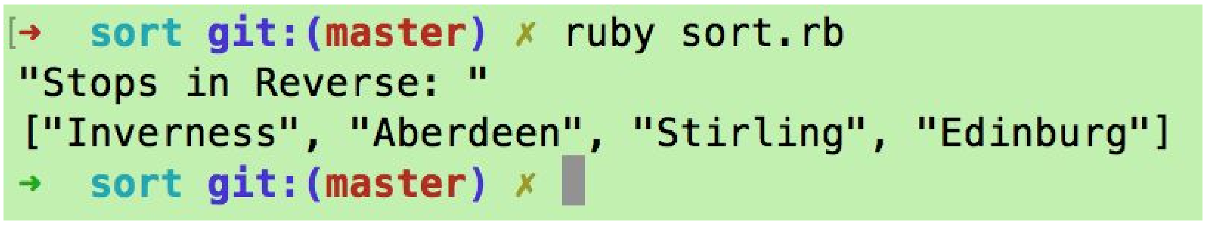
I.T 4 - Demonstrate sorting data in a program. Take screenshots of:

- A function that **sorts data**:



```ruby
@stops = [ "Edinburg", "Stirling", "Aberdeen", "Inverness" ]

# Reverse and print the positions of the stops in the array
def reverse_and_print_stops
  p "Stops in Reverse: "
  p @stops.reverse!()
end

reverse_and_print_stops
```

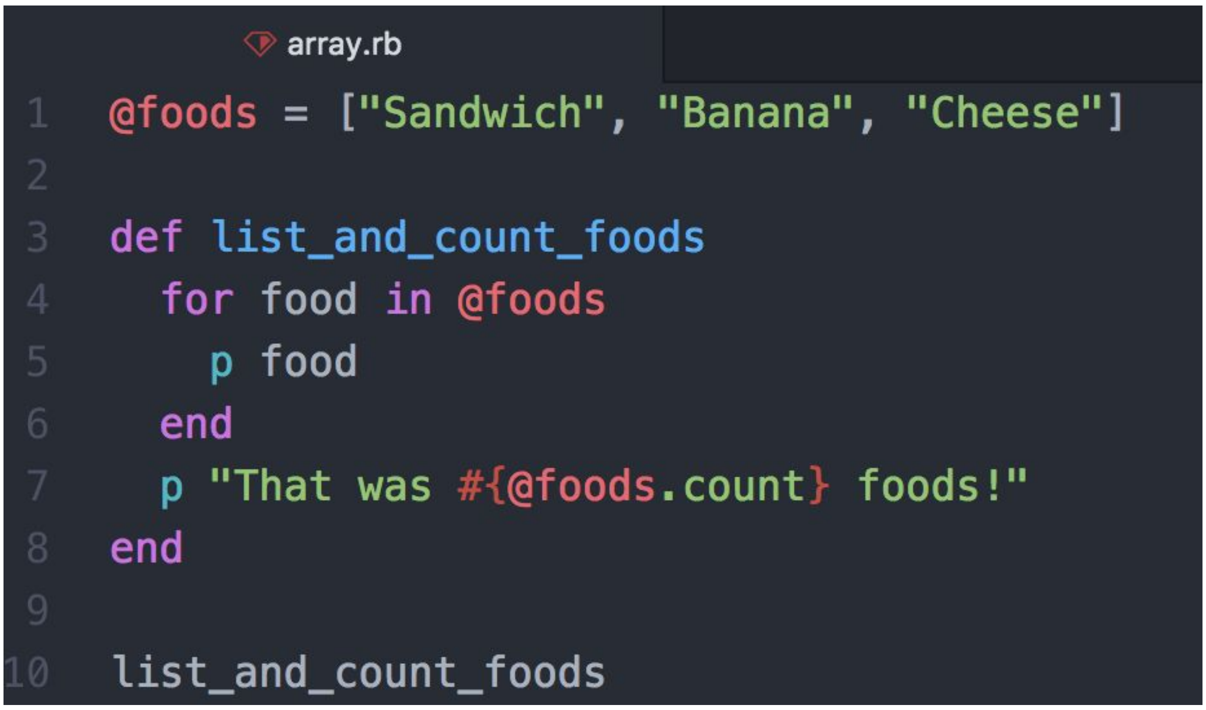
- The result of the function running:



```
[→  sort git:(master) ✗ ruby sort.rb
"Stops in Reverse: "
["Inverness", "Aberdeen", "Stirling", "Edinburg"]
→  sort git:(master) ✗ 
```

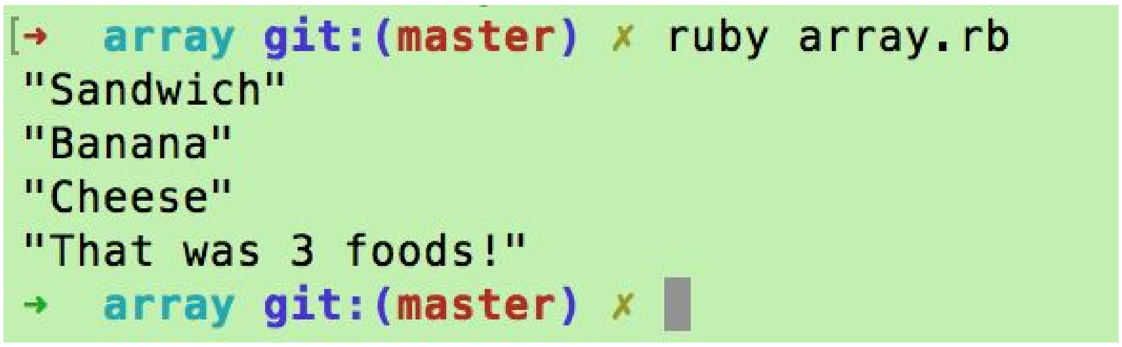
I.T 5 - Demonstrate the use of an array in a program. Take screenshots of:

- An **Array** in a program + A function that uses an **array**:



```ruby
array.rb
@foods = ["Sandwich", "Banana", "Cheese"]

def list_and_count_foods
  for food in @foods
    p food
  end
  p "That was #{@foods.count} foods!"
end

list_and_count_foods
```

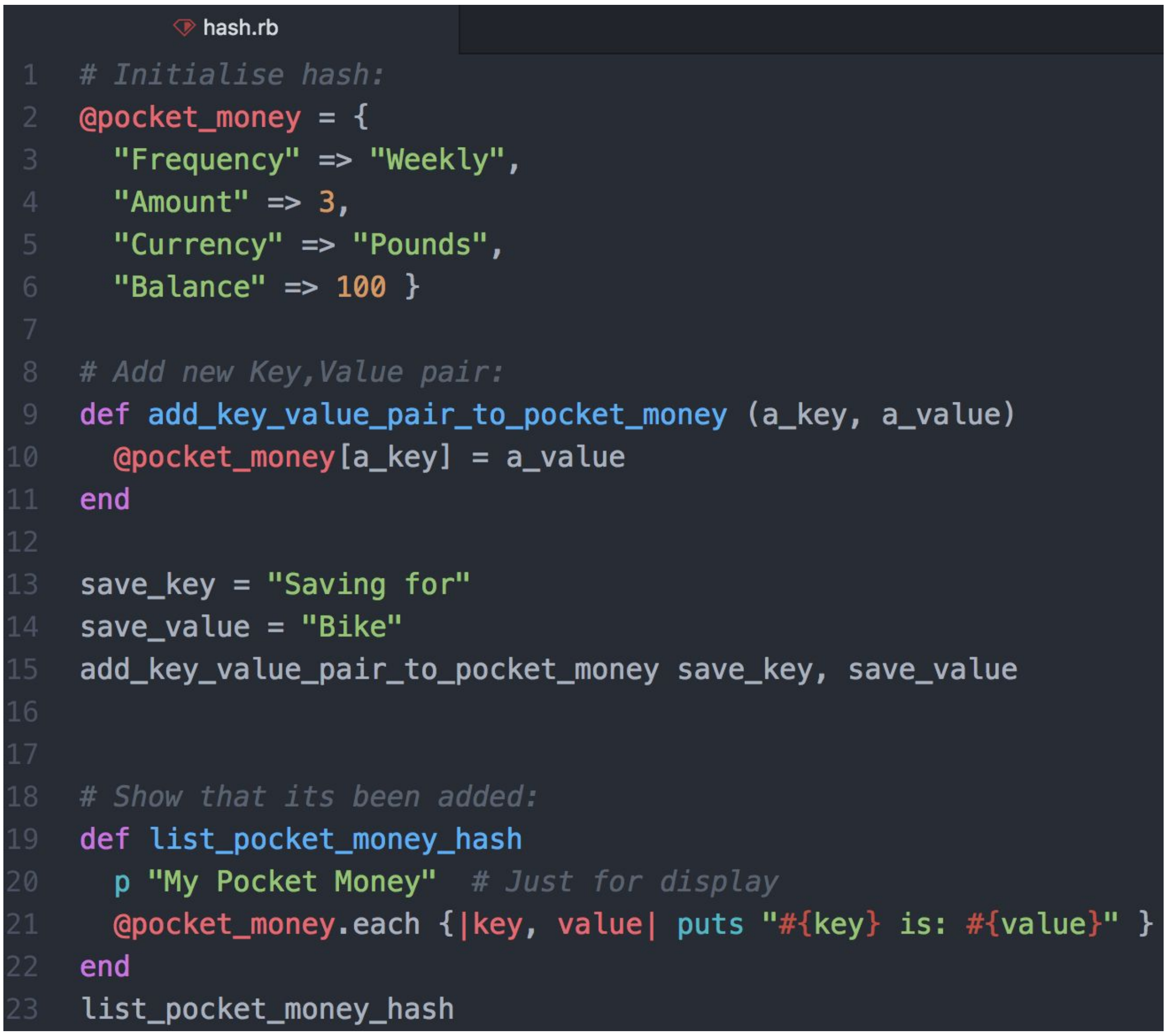
- The result of the function running:



```
[➜  array git:(master) ✗ ruby array.rb
"Sandwich"
"Banana"
"Cheese"
"That was 3 foods!"
➜  array git:(master) ✗ 
```

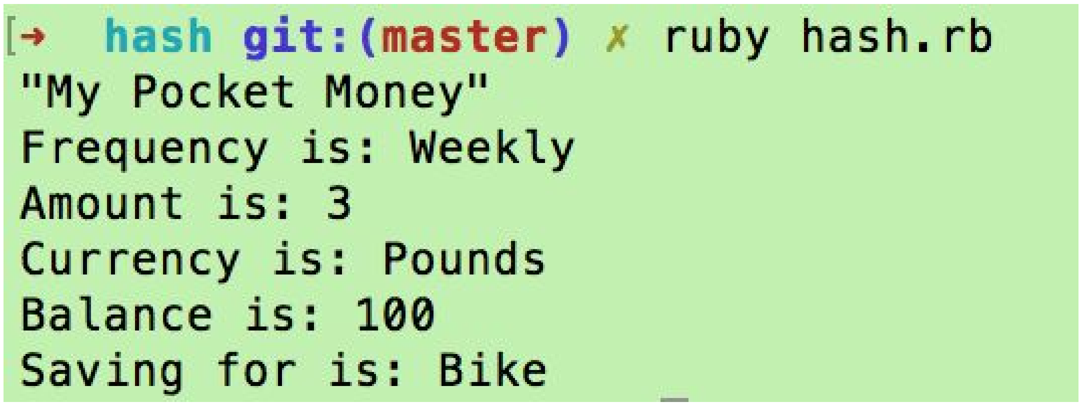
I.T 6 Demonstrate searching data in a program. Take screenshots of:

- A **hash** in a program + A function that uses a **hash**:



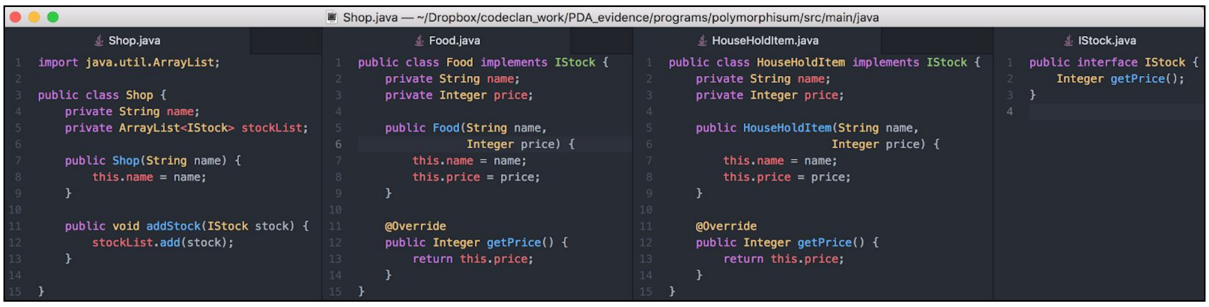
```ruby
# Initialise hash:
@pocket_money = {
  "Frequency" => "Weekly",
  "Amount" => 3,
  "Currency" => "Pounds",
  "Balance" => 100 }

# Add new Key,Value pair:
def add_key_value_pair_to_pocket_money (a_key, a_value)
  @pocket_money[a_key] = a_value
end

save_key = "Saving for"
save_value = "Bike"
add_key_value_pair_to_pocket_money save_key, save_value


# Show that its been added:
def list_pocket_money_hash
  p "My Pocket Money"  # Just for display
  @pocket_money.each {|key, value| puts "#{key} is: #{value}" }
end
list_pocket_money_hash
```

- The result of the function running:



```
[→  hash git:(master) ✗ ruby hash.rb
"My Pocket Money"
Frequency is: Weekly
Amount is: 3
Currency is: Pounds
Balance is: 100
Saving for is: Bike
```

I.T 7 - The use of Polymorphism in a program:


Shop.java — ~/Dropbox/codeclan_work/PDA_evidence/programs/polymorphisum/src/main/java

Shop.java
```java
import java.util.ArrayList;

public class Shop {
    private String name;
    private ArrayList<IStock> stockList;

    public Shop(String name) {
        this.name = name;
    }

    public void addStock(IStock stock) {
        stockList.add(stock);
    }

}
```

Food.java
```java
public class Food implements IStock {
    private String name;
    private Integer price;

    public Food(String name,
                      Integer price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public Integer getPrice() {
        return this.price;
    }
}
```

HouseHoldItem.java
```java
public class HouseHoldItem implements IStock {
    private String name;
    private Integer price;

    public HouseHoldItem(String name,
                              Integer price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public Integer getPrice() {
        return this.price;
    }
}
```

IStock.java
```java
public interface IStock {
    Integer getPrice();
}

```

Interface *IStock* has method getPrice which returns an Integer.

Food and *HouseHoldItem* classes both implement interface *IStock*.

Shop class has ArrayList of items *IStock* called *stockList*.

This ArrayList can take both *Food* and *HouseHoldItem* objects which are different classes because it treats them both as *IStock* objects. This is an example of polymorphism.