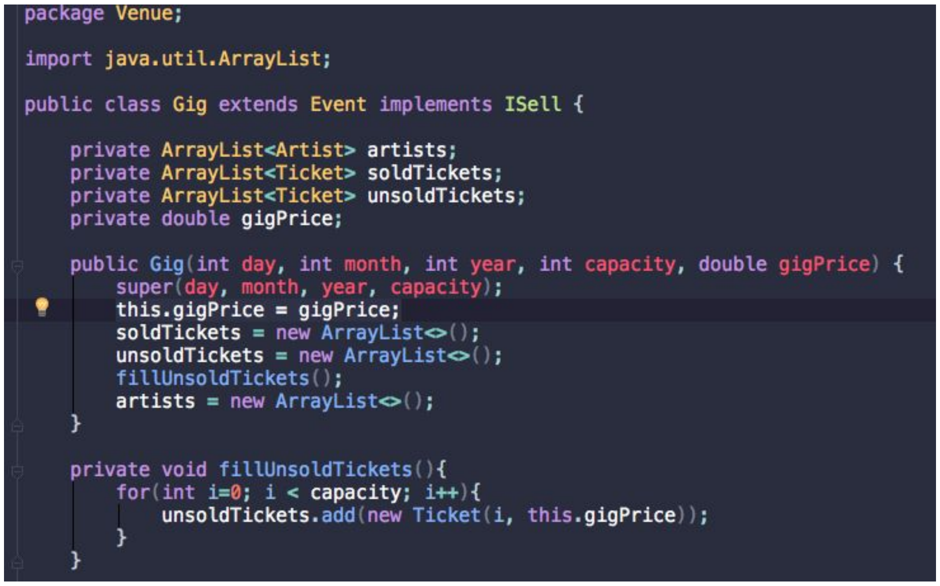
Evidence for Implementation and Testing Unit

Catriona Meriel
E17

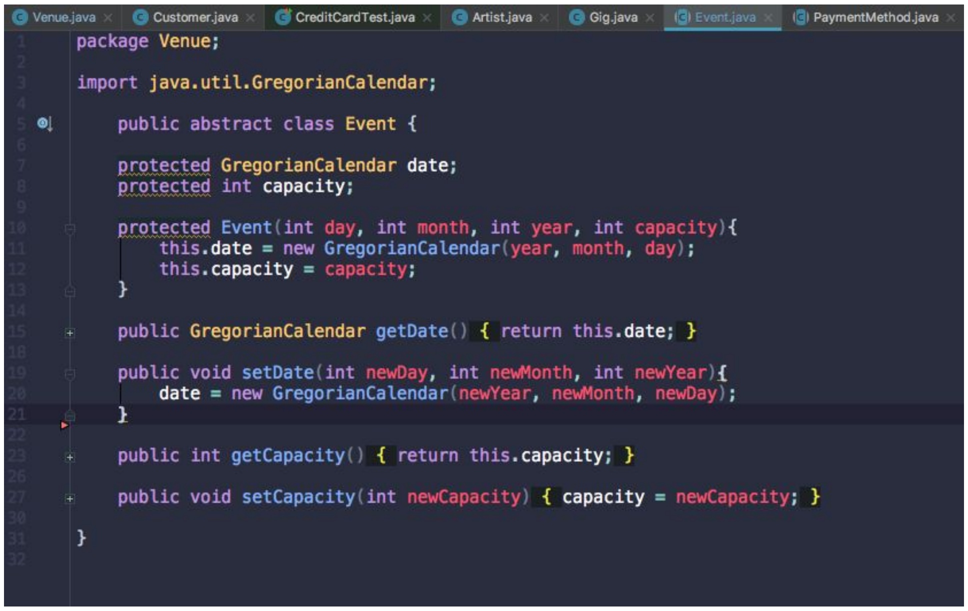
I.T 1 - Demonstrate one example of encapsulation that you have written in a program



```java
package Venue;

import java.util.ArrayList;

public class Gig extends Event implements ISell {

    private ArrayList<Artist> artists;
    private ArrayList<Ticket> soldTickets;
    private ArrayList<Ticket> unsoldTickets;
    private double gigPrice;

    public Gig(int day, int month, int year, int capacity, double gigPrice) {
        super(day, month, year, capacity);
        this.gigPrice = gigPrice;
        soldTickets = new ArrayList<>();
        unsoldTickets = new ArrayList<>();
        fillUnsoldTickets();
        artists = new ArrayList<>();
    }

    private void fillUnsoldTickets(){
        for(int i=0; i < capacity; i++){
            unsoldTickets.add(new Ticket(i, this.gigPrice));
        }
    }
```

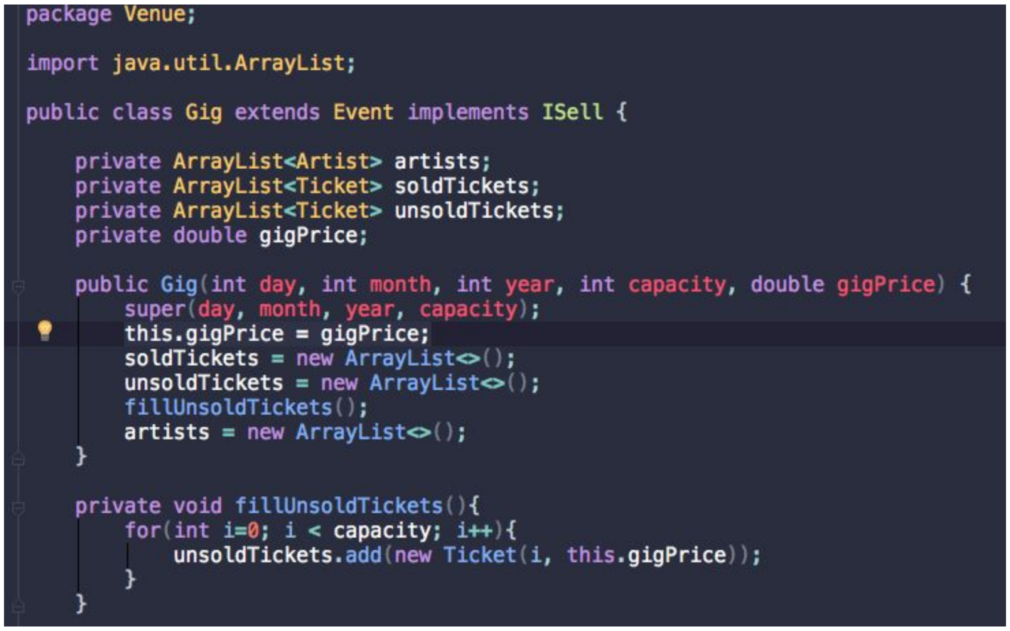
Shows property encapsulation and method encapsulation

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

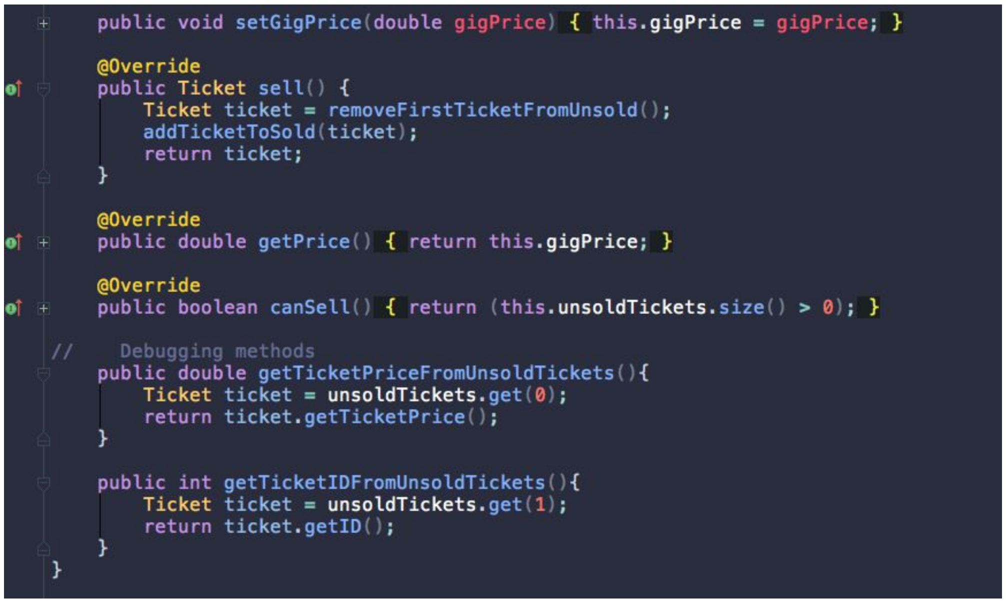


```java
package Venue;

import java.util.GregorianCalendar;

    public abstract class Event {

    protected GregorianCalendar date;
    protected int capacity;

    protected Event(int day, int month, int year, int capacity){
        this.date = new GregorianCalendar(year, month, day);
        this.capacity = capacity;
    }

    public GregorianCalendar getDate() { return this.date; }

    public void setDate(int newDay, int newMonth, int newYear){
        date = new GregorianCalendar(newYear, newMonth, newDay);
    }

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

    public void setCapacity(int newCapacity) { capacity = newCapacity; }

}
```

Event is an abstract parent class

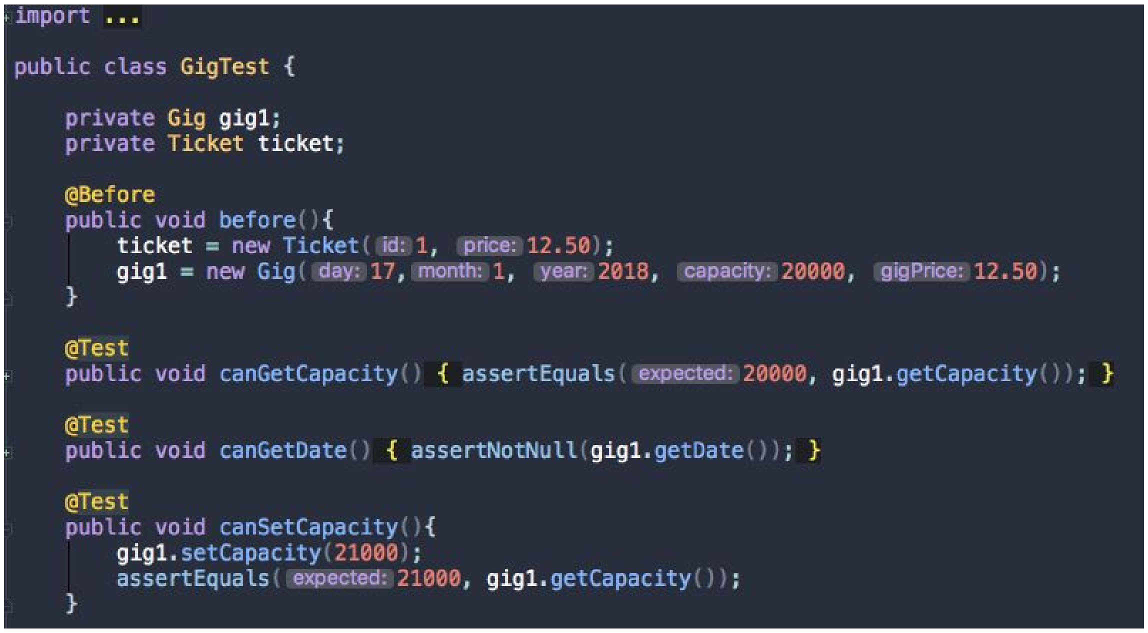


```java
package Venue;

import java.util.ArrayList;

public class Gig extends Event implements ISell {

    private ArrayList<Artist> artists;
    private ArrayList<Ticket> soldTickets;
    private ArrayList<Ticket> unsoldTickets;
    private double gigPrice;

    public Gig(int day, int month, int year, int capacity, double gigPrice) {
        super(day, month, year, capacity);
        this.gigPrice = gigPrice;
        soldTickets = new ArrayList<>();
        unsoldTickets = new ArrayList<>();
        fillUnsoldTickets();
        artists = new ArrayList<>();
    }

    private void fillUnsoldTickets(){
        for(int i=0; i < capacity; i++){
            unsoldTickets.add(new Ticket(i, this.gigPrice));
        }
    }
```



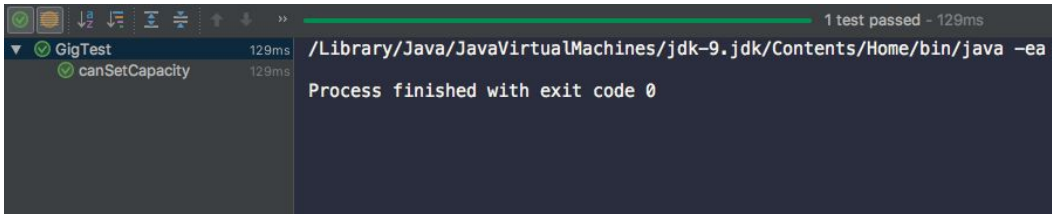
```java
    public void setGigPrice(double gigPrice) { this.gigPrice = gigPrice; }

    @Override
    public Ticket sell() {
        Ticket ticket = removeFirstTicketFromUnsold();
        addTicketToSold(ticket);
        return ticket;
    }

    @Override
    public double getPrice() { return this.gigPrice; }

    @Override
    public boolean canSell() { return (this.unsoldTickets.size() > 0); }

//    Debugging methods
    public double getTicketPriceFromUnsoldTickets(){
        Ticket ticket = unsoldTickets.get(0);
        return ticket.getTicketPrice();
    }

    public int getTicketIDFromUnsoldTickets(){
        Ticket ticket = unsoldTickets.get(1);
        return ticket.getID();
    }
}
```

Gig inherits from Event



```java
import ...

public class GigTest {

    private Gig gig1;
    private Ticket ticket;

    @Before
    public void before(){
        ticket = new Ticket( id: 1,  price: 12.50);
        gig1 = new Gig( day: 17, month: 1,  year: 2018,  capacity: 20000,  gigPrice: 12.50);
    }

    @Test
    public void canGetCapacity() { assertEquals( expected: 20000, gig1.getCapacity()); }

    @Test
    public void canGetDate() { assertNotNull(gig1.getDate()); }

    @Test
    public void canSetCapacity(){
        gig1.setCapacity(21000);
        assertEquals( expected: 21000, gig1.getCapacity());
    }
```

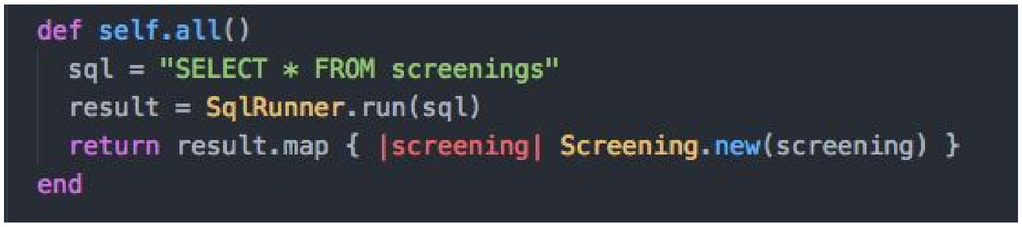
gig1 is an instance of Gig and uses the method from Event but on the child class Gig



```
1 test passed - 129ms
GigTest             129ms
  canSetCapacity    129ms

/Library/Java/JavaVirtualMachines/jdk-9.jdk/Contents/Home/bin/java -ea

Process finished with exit code 0
```

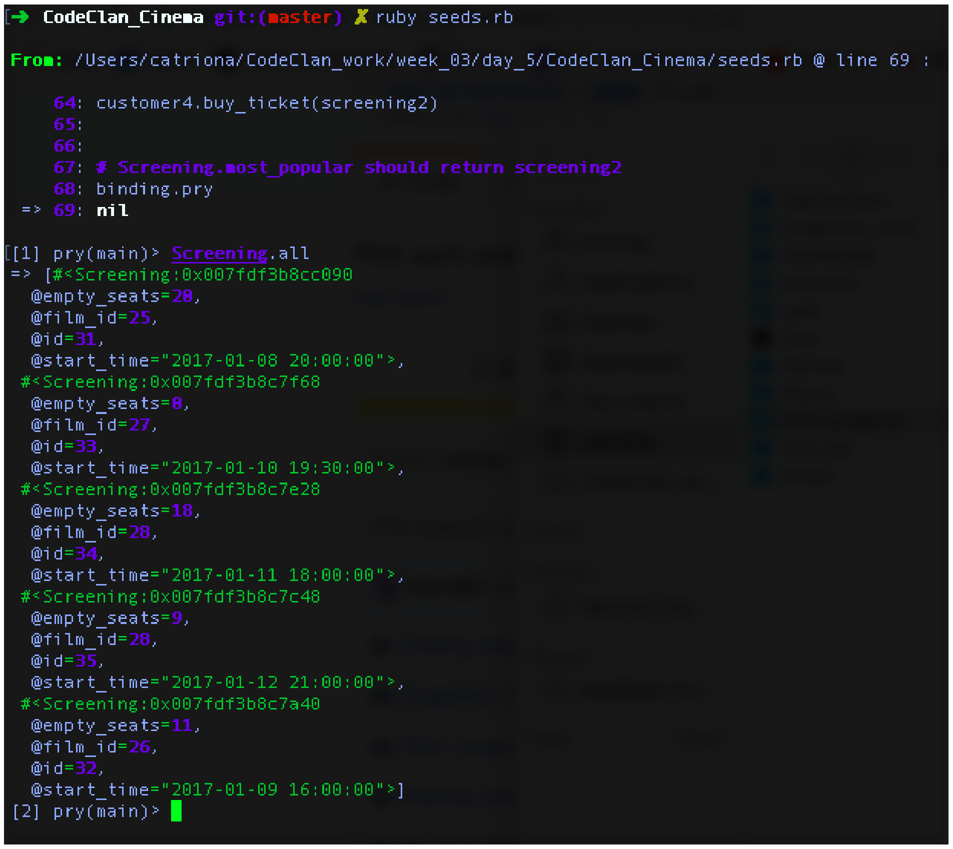
Test passes

I.T 3 - Example of searching



```ruby
def self.all()
  sql = "SELECT * FROM screenings"
  result = SqlRunner.run(sql)
  return result.map { |screening| Screening.new(screening) }
end
```

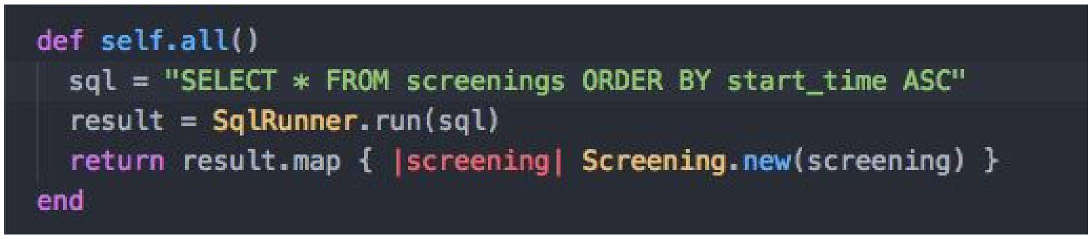
Function which searches for all the screenings



```
[➜ CodeClan_Cinema git:(master) ✗ ruby seeds.rb

From: /Users/catriona/CodeClan_work/week_03/day_5/CodeClan_Cinema/seeds.rb @ line 69 :

    64: customer4.buy_ticket(screening2)
    65:
    66:
    67: # Screening.most_popular should return screening2
    68: binding.pry
 => 69: nil

[1] pry(main)> Screening.all
=> [#<Screening:0x007fdf3b8cc090
  @empty_seats=20,
  @film_id=25,
  @id=31,
  @start_time="2017-01-08 20:00:00">,
 #<Screening:0x007fdf3b8c7f68
  @empty_seats=8,
  @film_id=27,
  @id=33,
  @start_time="2017-01-10 19:30:00">,
 #<Screening:0x007fdf3b8c7e28
  @empty_seats=18,
  @film_id=28,
  @id=34,
  @start_time="2017-01-11 18:00:00">,
 #<Screening:0x007fdf3b8c7c48
  @empty_seats=9,
  @film_id=28,
  @id=35,
  @start_time="2017-01-12 21:00:00">,
 #<Screening:0x007fdf3b8c7a40
  @empty_seats=11,
  @film_id=26,
  @id=32,
  @start_time="2017-01-09 16:00:00">]
[2] pry(main)>
```

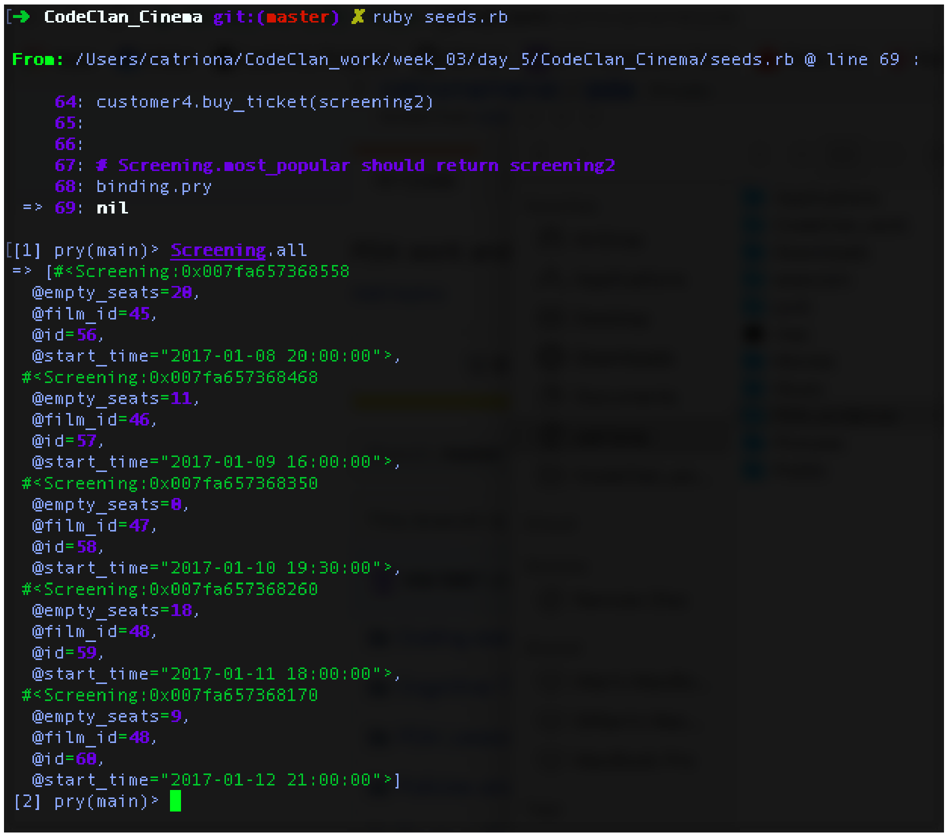
The result from the search

I.T 4 – Example of sorting



```ruby
def self.all()
  sql = "SELECT * FROM screenings ORDER BY start_time ASC"
  result = SqlRunner.run(sql)
  return result.map { |screening| Screening.new(screening) }
end
```

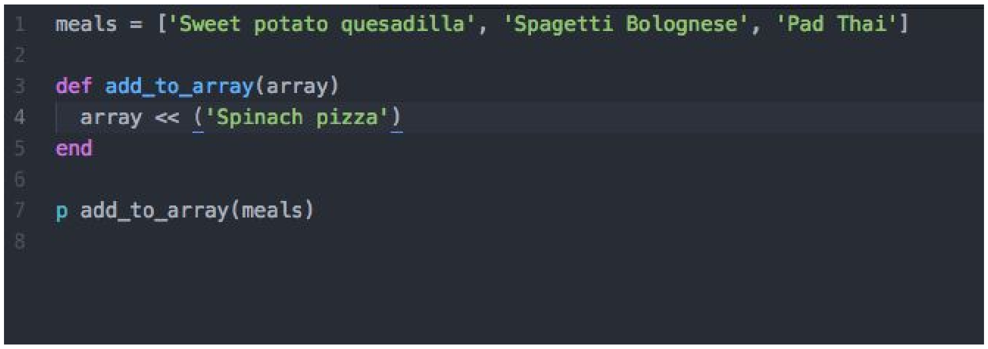
Sorting all the screenings by start time in ascending order



```
[➜ CodeClan_Cinema git:(master) ✗ ruby seeds.rb

From: /Users/catriona/CodeClan_work/week_03/day_5/CodeClan_Cinema/seeds.rb @ line 69 :

    64: customer4.buy_ticket(screening2)
    65:
    66:
    67: # Screening.most_popular should return screening2
    68: binding.pry
 => 69: nil

[1] pry(main)> Screening.all
=> [#<Screening:0x007fa657368558
  @empty_seats=20,
  @film_id=45,
  @id=56,
  @start_time="2017-01-08 20:00:00">,
 #<Screening:0x007fa657368468
  @empty_seats=11,
  @film_id=46,
  @id=57,
  @start_time="2017-01-09 16:00:00">,
 #<Screening:0x007fa657368350
  @empty_seats=8,
  @film_id=47,
  @id=58,
  @start_time="2017-01-10 19:30:00">,
 #<Screening:0x007fa657368260
  @empty_seats=18,
  @film_id=48,
  @id=59,
  @start_time="2017-01-11 18:00:00">,
 #<Screening:0x007fa657368170
  @empty_seats=9,
  @film_id=48,
  @id=60,
  @start_time="2017-01-12 21:00:00">]
[2] pry(main)>
```

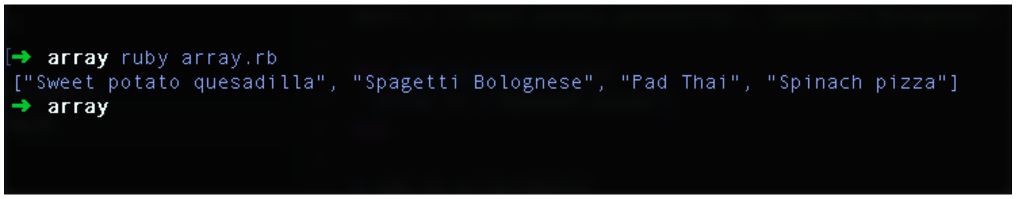
Result of sort

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



```ruby
meals = ['Sweet potato quesadilla', 'Spagetti Bolognese', 'Pad Thai']

def add_to_array(array)
  array << ('Spinach pizza')
end

p add_to_array(meals)

```

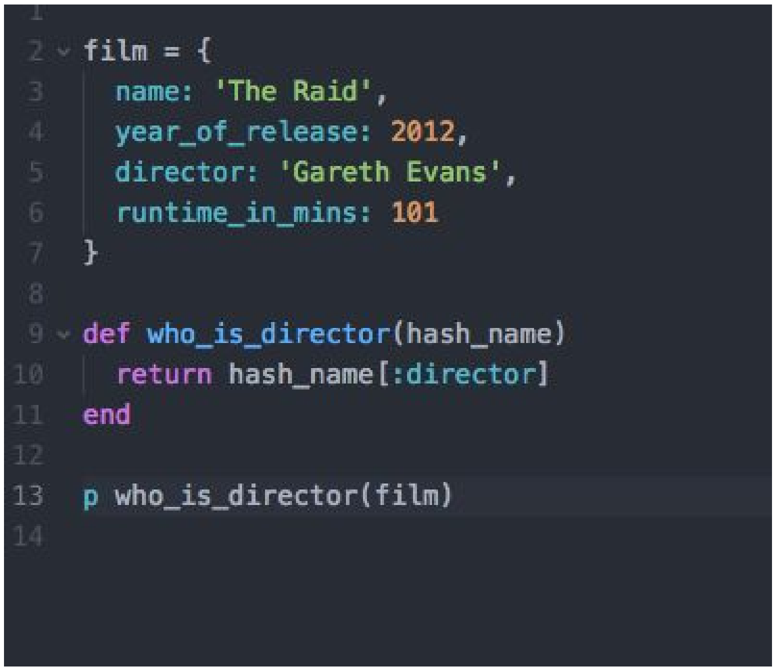
A meals array and a function which adds to this array



```
[➜ array ruby array.rb
["Sweet potato quesadilla", "Spagetti Bolognese", "Pad Thai", "Spinach pizza"]
➜ array
```

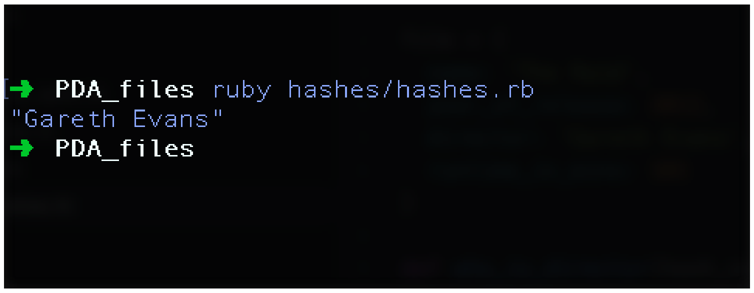
Result of adding a new meal to the array

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



```ruby
film = {
  name: 'The Raid',
  year_of_release: 2012,
  director: 'Gareth Evans',
  runtime_in_mins: 101
}

def who_is_director(hash_name)
  return hash_name[:director]
end

p who_is_director(film)
```

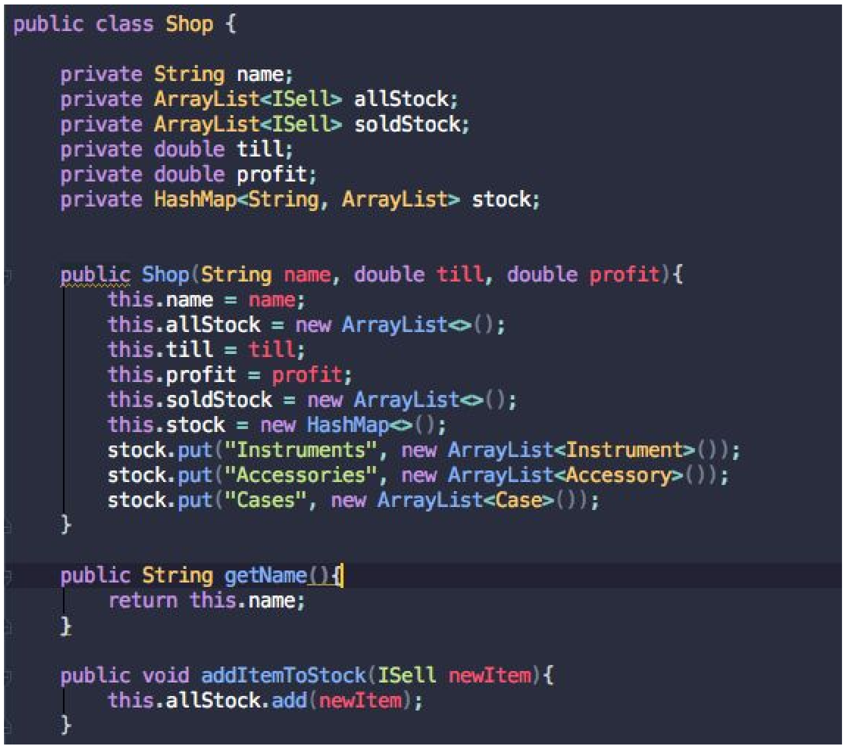
A hash containing the details of a film and a function which returns the director of the film



```
➜ PDA_files ruby hashes/hashes.rb
"Gareth Evans"
➜ PDA_files
```

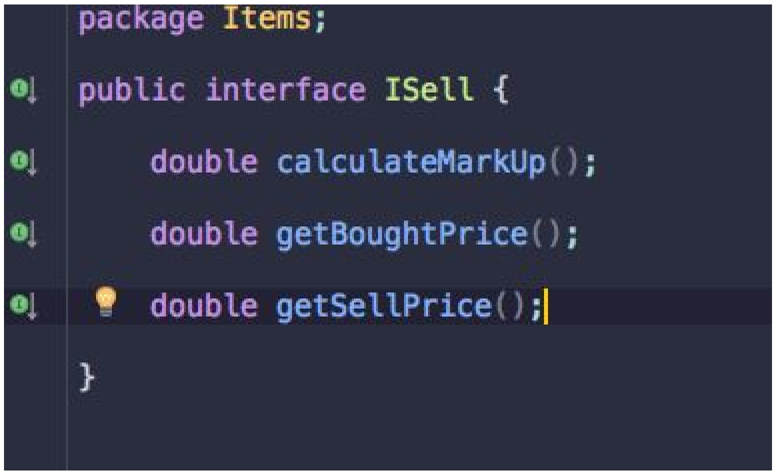
Result of function being called

I.T 7 - Example of polymorphism in a program



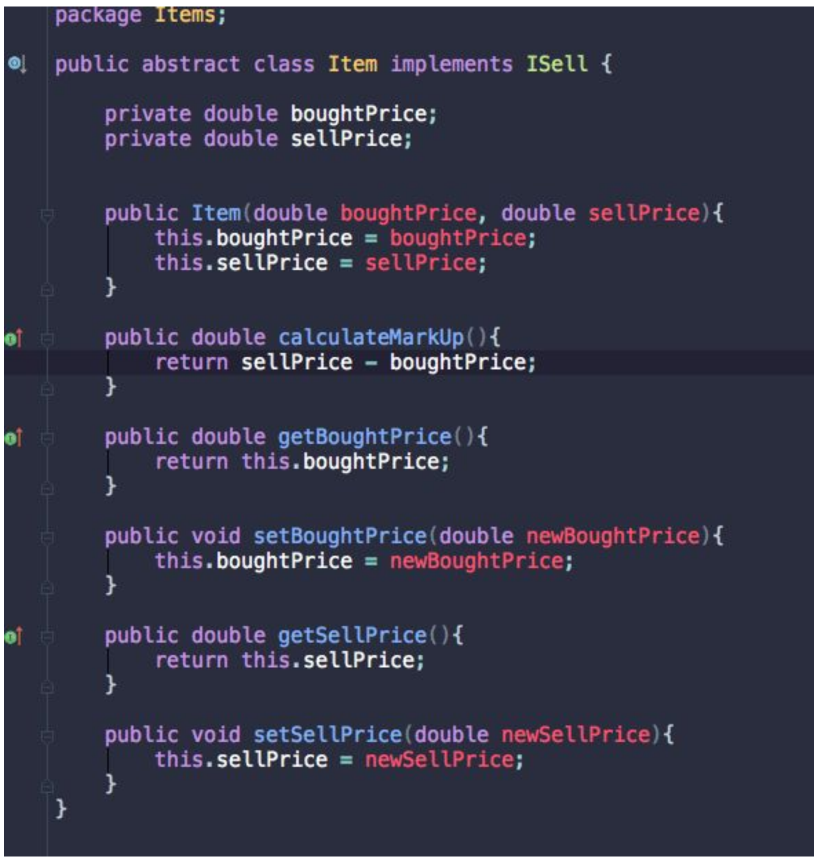
```java
public class Shop {

    private String name;
    private ArrayList<ISell> allStock;
    private ArrayList<ISell> soldStock;
    private double till;
    private double profit;
    private HashMap<String, ArrayList> stock;


    public Shop(String name, double till, double profit){
        this.name = name;
        this.allStock = new ArrayList<>();
        this.till = till;
        this.profit = profit;
        this.soldStock = new ArrayList<>();
        this.stock = new HashMap<>();
        stock.put("Instruments", new ArrayList<Instrument>());
        stock.put("Accessories", new ArrayList<Accessory>());
        stock.put("Cases", new ArrayList<Case>());
    }

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

    public void addItemToStock(ISell newItem){
        this.allStock.add(newItem);
    }
```

An example of a shop class using polymorphism. It's list of stock can be any class that implements the ISell interface



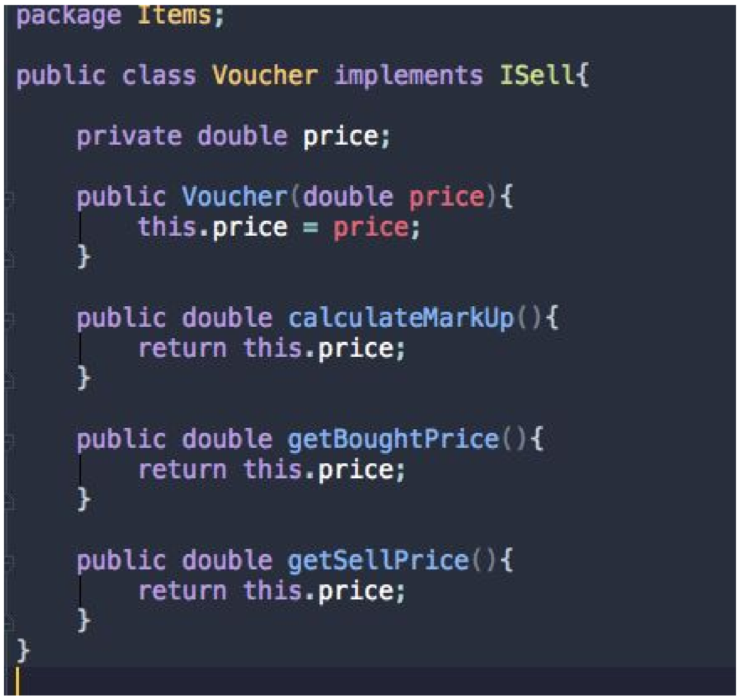
```java
package Items;

public interface ISell {

    double calculateMarkUp();

    double getBoughtPrice();

    double getSellPrice();

}
```

Interface ISell



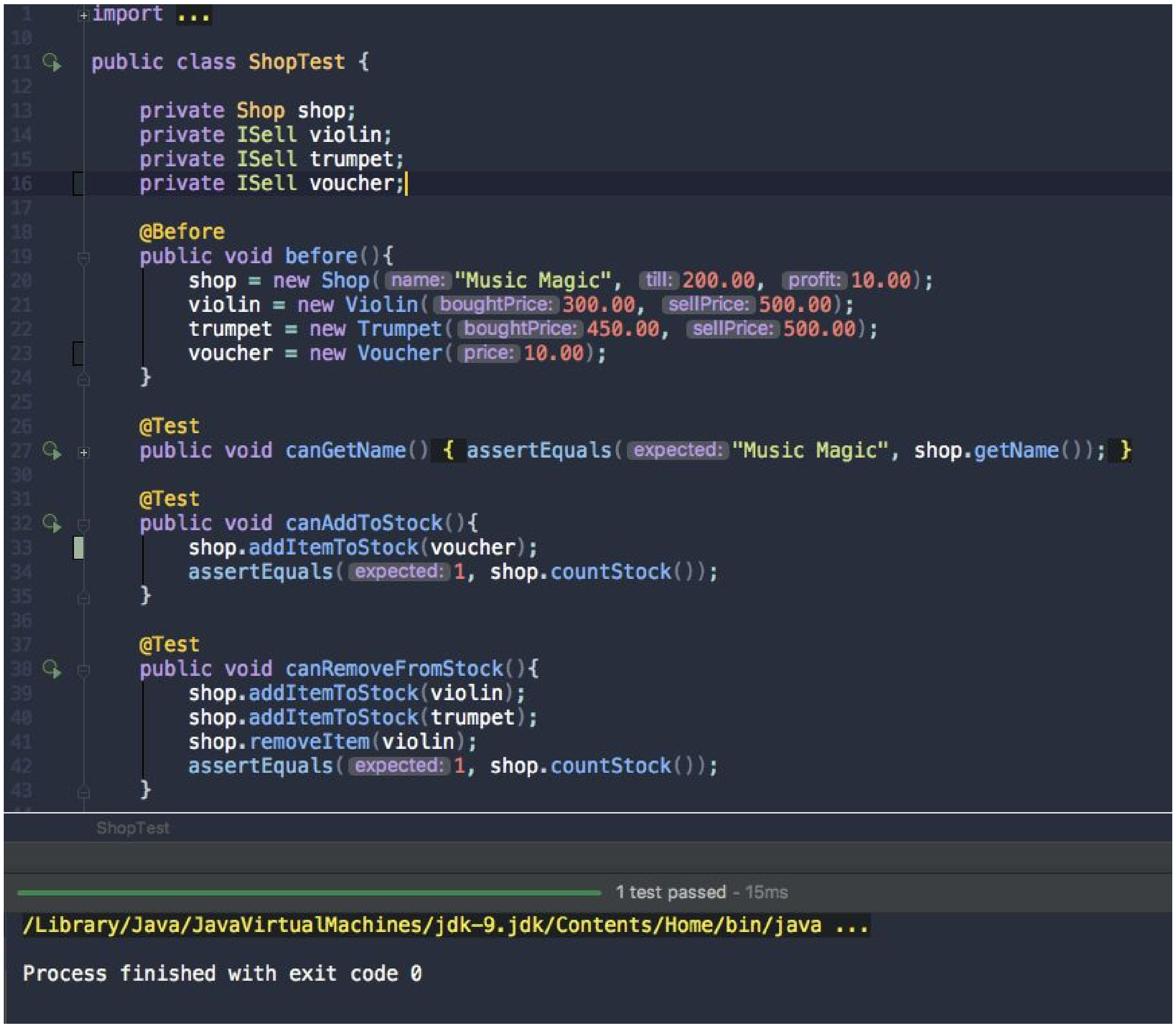
```java
package Items;

public abstract class Item implements ISell {

    private double boughtPrice;
    private double sellPrice;


    public Item(double boughtPrice, double sellPrice){
        this.boughtPrice = boughtPrice;
        this.sellPrice = sellPrice;
    }

    public double calculateMarkUp(){
        return sellPrice - boughtPrice;
    }

    public double getBoughtPrice(){
        return this.boughtPrice;
    }

    public void setBoughtPrice(double newBoughtPrice){
        this.boughtPrice = newBoughtPrice;
    }

    public double getSellPrice(){
        return this.sellPrice;
    }

    public void setSellPrice(double newSellPrice){
        this.sellPrice = newSellPrice;
    }
}
```

Abstract class Item implementing interface ISell



```java
package Items;

public class Voucher implements ISell{

    private double price;

    public Voucher(double price){
        this.price = price;
    }

    public double calculateMarkUp(){
        return this.price;
    }

    public double getBoughtPrice(){
        return this.price;
    }

    public double getSellPrice(){
        return this.price;
    }
}
```

Class Voucher implementing interface ISell



```java
import ...

public class ShopTest {

    private Shop shop;
    private ISell violin;
    private ISell trumpet;
    private ISell voucher;

    @Before
    public void before(){
        shop = new Shop( name: "Music Magic",  till: 200.00,  profit: 10.00);
        violin = new Violin( boughtPrice: 300.00,  sellPrice: 500.00);
        trumpet = new Trumpet( boughtPrice: 450.00,  sellPrice: 500.00);
        voucher = new Voucher( price: 10.00);
    }

    @Test
    public void canGetName() { assertEquals( expected: "Music Magic", shop.getName()); }

    @Test
    public void canAddToStock(){
        shop.addItemToStock(voucher);
        assertEquals( expected: 1, shop.countStock());
    }

    @Test
    public void canRemoveFromStock(){
        shop.addItemToStock(violin);
        shop.addItemToStock(trumpet);
        shop.removeItem(violin);
        assertEquals( expected: 1, shop.countStock());
    }
```

ShopTest

1 test passed - 15ms

```
/Library/Java/JavaVirtualMachines/jdk-9.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

This shows the Shop tests passing when a new ISell object (voucher or violin) is added and removed from the stock