I.T 1

-encapsulation in a program

```java
public abstract class Meal implements Serializable{

    private String food;
    private int calories;

    public Meal(String food, int calories){
        this.food = food;
        this.calories = calories;
    }

    public String getFood() { return this.food; }

    public int getCalories() { return this.calories; }

    public void setFood(String food) { this.food = food; }

    public void setCalories(int calories) { this.calories = calories; }
}
```

I.T 2

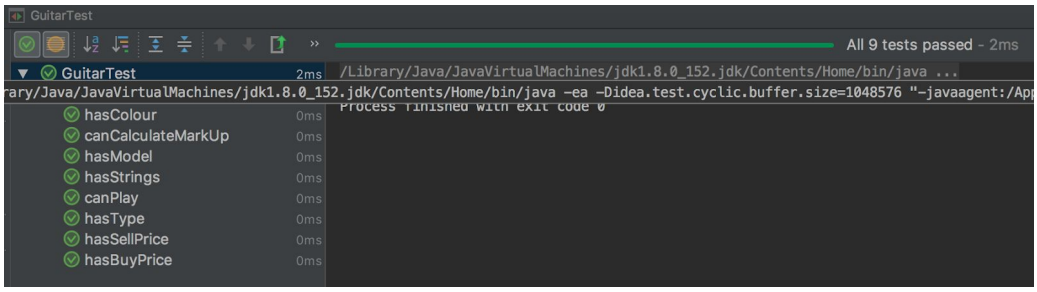
-A Class

```java
public abstract class Instrument implements IPlayable, ISellable{

    private Type type;
    private String model;
    private String colour;
    private String material;
    private double buyPrice;
    private double sellPrice;

    public Instrument(Type type, String model, String colour, String material, double buyPrice, double sellPrice){
        this.type = type;
        this.model = model;
        this.colour = colour;
        this.material = material;
        this.buyPrice = buyPrice;
        this.sellPrice = sellPrice;
    }


    public Type getType() {
        return this.type;
    }

    public String getModel() {
        return this.model;
    }

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

-A Class that inherits from the previous class

```java
public class Guitar extends Instrument{

    private int strings;

    public Guitar(Type type, String model, String colour, String material, double buyPrice, double sellPrice, int strings) {
        super(type, model, colour, material, buyPrice, sellPrice);
        this.strings = strings;
    }


    public int getStrings() {
        return this.strings;
    }

    public String play(){
        return "Strumming away";
    }
}
```

```java
public class GuitarTest {

    Guitar guitar;

    @Before
    public void before(){
        guitar = new Guitar(Type.STRING,  model: "Yamaha GL1",  colour: "Brown",  material: "Wood",  buyPrice: 44,  sellPrice: 62,  strings: 6);
    }

    @Test
    public void hasType(){
        assertEquals(Type.STRING, guitar.getType());
    }

    @Test
    public void hasModel(){
        assertEquals( expected: "Yamaha GL1", guitar.getModel());
    }

    @Test
    public void hasColour(){
        assertEquals( expected: "Brown", guitar.getColour());
    }
```



I. T 3

Function that searches data:

```ruby
def Transaction.find(id)
  sql = "SELECT * FROM transactions
        WHERE id = $1;"
  values = [id]
  transaction_hash = SqlRunner.run(sql, values).first
  return Transaction.new(transaction_hash)
end
```

The result of the function running:

```
[1] pry(main)> Transaction.find(1)
=> #<Transaction:0x007ff83c25b200
 @amount=10.0,
 @category_id=1,
 @comment="Lost so went to MacDonalds!",
 @id=1,
 @transaction_date="2017-12-19",
 @vendor_id=1>
```

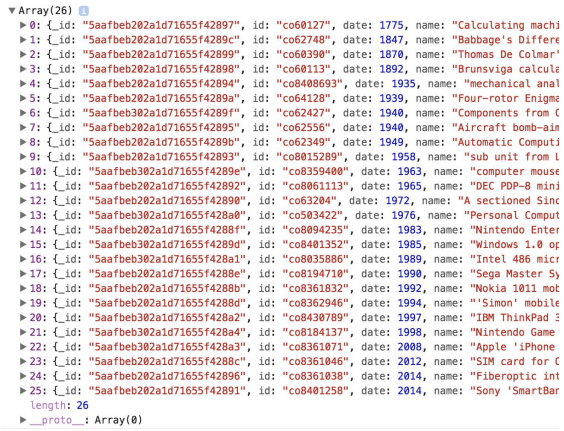
I. T 4

Function that sorts data:

```
ComputerObjectView.prototype.sortByDate = function () {
  this.computerObjects.sort(function (earliest, latest) {
    return earliest.date - latest.date;
  });
}
```

```
computerObjectView.sortByDate();
console.log(computerObjectView.computerObjects);
```

The result of the function running:



I. T 5

An array in a program:

```
@customers = [
  {
    name: "Craig",
    pets: [],
    cash: 1000
  },
  {
    name: "Zsolt",
    pets: [],
    cash: 50
  }
]
```

A function that uses the array:

```
def customer_pet_count(customer)
  return customer[:pets].count
end
```

The result of the function running:

```ruby
def test_customer_pet_count
  count = customer_pet_count(@customers[0])
  assert_equal(0, count)
  p count
end
```

```
➜  week_01_pet_shop git:(master) ✗ ruby specs/pet_shop_spec.rb
Run options: --seed 54931

# Running:

..............0
....

Finished in 0.001510s, 13245.0344 runs/s, 17218.5447 assertions/s.

20 runs, 26 assertions, 0 failures, 0 errors, 0 skips
```

I. T 6

A hash in a program:

```ruby
@pet_shop = {
    pets: [
      {
        name: "Sir Percy",
        pet_type: :cat,
        breed: "British Shorthair",
        price: 500
      },
      {
        name: "King Bagdemagus",
        pet_type: :cat,
        breed: "British Shorthair",
        price: 500
      },
      {
        name: "Sir Lancelot",
        pet_type: :dog,
        breed: "Pomsky",
        price: 1000,
      },
      {
        name: "Arthur",
        pet_type: :dog,
        breed: "Husky",
        price: 900,
      },
      {
        name: "Tristan",
        pet_type: :dog,
        breed: "Basset Hound",
        price: 800,
      },
      {
        name: "Merlin",
        pet_type: :cat,
        breed: "Egyptian Mau",
        price: 1500,
      }
    ],
    admin: {
      total_cash: 1000,
      pets_sold: 0,
    },
    name: "Camelot of Pets"
  }
```

A function that uses the hash:

```ruby
def pet_shop_name(pet_shop)
  return pet_shop[:name]
end
```

The result of the function running:

```ruby
def test_pet_shop_name
  name = pet_shop_name(@pet_shop)
  assert_equal("Camelot of Pets", name)
  p name
end
```

```
[➜ week_01_pet_shop git:(master) × ruby specs/pet_shop_spec.rb ]
Run options: --seed 38346

# Running:

......"Camelot of Pets"
..............

Finished in 0.001464s, 13661.2024 runs/s, 17759.5632 assertions/s.

20 runs, 26 assertions, 0 failures, 0 errors, 0 skips
```

I.T 7

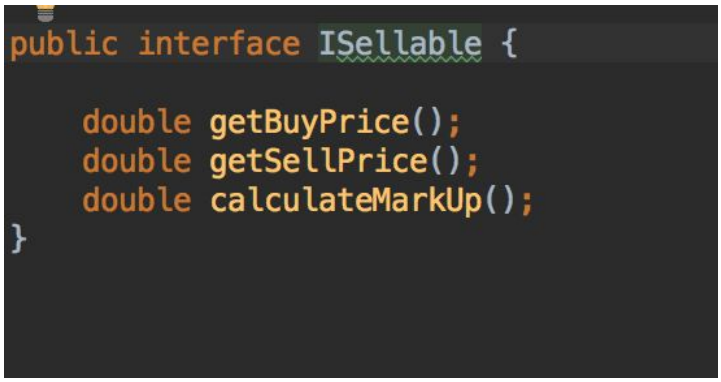
-use of polymorphism in a program

```java
public class MusicShop {

    private String name;
    private ArrayList<ISellable> stock;
    private double budget;

    public MusicShop(String name, double budget){
        this.name = name;
        this.stock = new ArrayList<>();
        this.budget = budget;
    }

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

    public int stockCount() {
        return this.stock.size();
    }

    public void addToStock(ISellable item) {
        if(item.getBuyPrice() <= getBudget()) {
            this.stock.add(item);
            this.budget -= item.getBuyPrice();
        }
    }
```

```java
public abstract class Instrument implements IPlayable, ISellable{

    private Type type;
    private String model;
    private String colour;
    private String material;
    private double buyPrice;
    private double sellPrice;

    public Instrument(Type type, String model, String colour, String material, double buyPrice, double sellPrice){
        this.type = type;
        this.model = model;
        this.colour = colour;
        this.material = material;
        this.buyPrice = buyPrice;
        this.sellPrice = sellPrice;
    }


    public Type getType() {
        return this.type;
    }

    public String getModel() { return this.model; }

    public String getColour() {
        return this.colour;
    }

    public String getMaterial() {
        return this.material;
    }

    public double getBuyPrice() {
        return this.buyPrice;
    }

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

    public double calculateMarkUp(){
        return this.sellPrice - this.buyPrice;
    }
}
```

```java
public class Saxophone extends Instrument {

    String size;

    public Saxophone(Type type, String model, String colour, String material, double buyPrice, double sellPrice, String size) {
        super(type, model, colour, material, buyPrice, sellPrice);
        this.size = size;
    }


    public String getSize() { return this.size; }

    public String play() { return "Puffing away"; }
}
```

```java
public class Tshirt implements ISellable{

    private String size;
    private String description;
    private double buyPrice;
    private double sellPrice;

    public Tshirt(String size, String description, double buyPrice, double sellPrice){
        this.size = size;
        this.description = description;
        this.buyPrice = buyPrice;
        this.sellPrice = sellPrice;
    }

    public String getSize() { return this.size; }

    public String getDescription() { return this.description; }

    public double getBuyPrice() { return this.buyPrice; }

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

    public double calculateMarkUp() { return this.sellPrice - this.buyPrice; }
}
```


public interface ISellable {
double getBuyPrice();
double getSellPrice();
double calculateMarkUp();
}