Head First Object-Oriented Analysis and Design

INTRODUCTION

The longest journey begins with the first step!

And you should have already traveled a few miles from home!

(You should know something about object-oriented programming before starting this course.)

An object has…

State

- Changes over time

Behavior

- What the object does in response to messages

Identity

- What makes the object unique

S

Dave
Age: 32
Height: 6’ 2”

Brett
Age: 35
Height: 5’ 10”

Gary
Age: 61
Height: 5’ 8”

Behavior

Identity

One can use the object's address or object ID if all else fails to distinguish one object from another

Classes

Define the properties and behavior of objects

Can have behavior and properties that are defined in the class but are independent of the individual objects

Inheritance

Inheritance

Polymorphism

Different type of objects can respond to the same message

Ability of an object to take many forms

Polymorphism

Encapsulation

Encapsulation

Homework

Go through the Java tutorial on OOP concepts at:

http://www.ntu.edu.sg/home/ehchua/programming/java/j3b_oopinheritancepolymorphism.html

HFOOAD Chapter 1

A SIMPLE APPLICATION

We build software to solve problems.
People have problems.

Therefore, we build software for people.

Good software not only solves immediate problems, but it can be maintained and modified to address the inevitable changes that the customer will want.

Rick’s Guitars

Maintain a guitar inventory

Locate guitars for customers

What your predecessor built

Does this make any sense to you?

A simplified view

But there's a problem…

The Guitar class

```
public class Guitar {

  private String serialNumber, builder, model, type, backWood, topWood;
  private double price;

  public Guitar(String serialNumber, double price,
                String builder, String model, String type,
                String backWood, String topWood) {
    this.serialNumber = serialNumber;
    this.price = price;
    this.builder = builder;
    this.model = model;
    this.type = type;
    this.backWood = backWood;
    this.topWood = topWood;
  }
public String getSerialNumber() {return serialNumber;}
  public double getPrice() {return price;}
  public void setPrice(float newPrice) {
    this.price = newPrice;
  }
  public String getBuilder() {return builder;}
  public String getModel() {return model;}
  public String getType() {return type;}
  public String getBackWood() {return backWood;}
  public String getTopWood() {return topWood;}
}
```

The Inventory class

```
public class Inventory {
  private List guitars;
  public Inventory() { guitars = new LinkedList(); }
  public void addGuitar(String serialNumber, double price,
                        String builder, String model,
                        String type, String backWood, String topWood)
{
    Guitar guitar = new Guitar(serialNumber, price, builder,
                               model, type, backWood, topWood);
    guitars.add(guitar);
  }
  public Guitar getGuitar(String serialNumber) {
    for (Iterator i = guitars.iterator(); i.hasNext(); ) {
      Guitar guitar = (Guitar)i.next();
      if (guitar.getSerialNumber().equals(serialNumber)) {
        return guitar;
      }
    }
    return null;
  }
```

The Inventory class continued

```
public Guitar search(Guitar searchGuitar) {
    for (Iterator i = guitars.iterator(); i.hasNext(); ) {
      Guitar guitar = (Guitar)i.next();
      String builder = searchGuitar.getBuilder();
      if ((builder != null) && (!builder.equals("")) &&
          (!builder.equals(guitar.getBuilder())))
        continue;
      String model = searchGuitar.getModel();
      if ((model != null) && (!model.equals("")) &&
          (!model.equals(guitar.getModel())))
        continue;
      String type = searchGuitar.getType();
      if ((type != null) && (!searchGuitar.equals("")) &&
          (!type.equals(guitar.getType())))
        continue;
      String backWood = searchGuitar.getBackWood();
      if ((backWood != null) && (!backWood.equals("")) &&
          (!backWood.equals(guitar.getBackWood())))
        continue;
      String topWood = searchGuitar.getTopWood();
      if ((topWood != null) && (!topWood.equals("")) &&
          (!topWood.equals(guitar.getTopWood())))
        continue;
      return guitar;
    }
    return null;
  }
}
```

What can you do?

1. Case insensitive comparison of maker
2. All case insensitive string comparisons
3. Use constants for various makers
4. Talk to Rick and get the problem details

Three steps to great software

1. Make sure the software does what the customer wants
2. Apply good object-oriented principles
3. Strive for a maintainable, reusable design

Step 1: Talk to Rick

What questions will you ask Rick?

Questions for Rick

1. Do you only sell guitars?
2. How will you update the inventory?
3. How should a search for a guitar really work?
4. Do you need reports about inventory and sales?

Rick says

Customers don't always know all of the characteristics of the guitar they want.

There's often more than one guitar that matches the customer's needs.

Customers often look for a guitar in a specific price range

I need reports and other capabilities in my inventory, but my #1 problem is finding the right guitar for the customer.

To do list:

1. If there are guitars in stock that fit the customer's needs, always find them.
2. Take into account typing mistakes by the customer or make it impossible to enter erroneous data.

The impact of our changes

1 Change data entry forms and code

2 Update the inventory database

3 …

4

Rick says

Customers don't always know all of the characteristics of the guitar they want.

There's often more than one guitar that matches the customer's needs.

Customers often look for a guitar in a specific price range

I need reports and other capabilities in my inventory, but my #1 problem is finding the right guitar for the customer.

To do list:

1. If there are guitars in stock that fit the customer's needs, always find them.
2. Take into account typing mistakes by the customer or make it impossible to enter erroneous data.
3. Find ALL matching guitars.

We can return an array of Guitars.
Why not a List of Guitars?
Does it really matter?
Maybe an iterator?

Solution from the textbook

```
public List search(Guitar searchGuitar) {
    List matchingGuitars = new LinkedList();
    for (Iterator i = guitars.iterator(); i.hasNext(); ) {
      Guitar guitar = (Guitar)i.next();
      // Ignore serial number since that's unique
      // Ignore price since that's unique
      if (searchGuitar.getBuilder() != guitar.getBuilder())
        continue;
      String model = searchGuitar.getModel().toLowerCase();
      if ((model != null) && (!model.equals("")) &&
          (!model.equals(guitar.getModel().toLowerCase())))
        continue;
      if (searchGuitar.getType() != guitar.getType())
        continue;
      if (searchGuitar.getBackWood() != guitar.getBackWood())
        continue;
      if (searchGuitar.getTopWood() != guitar.getTopWood())
        continue;
      matchingGuitars.add(guitar);
    }
    return matchingGuitars;
  }
```

Alternate solution

```
public Iterator<Guitar> search(Guitar searchGuitar) {
    List<Guitar> matchingGuitars = new LinkedList<Guitar>();
    for (Iterator<Guitar> i = guitars.iterator(); i.hasNext(); ) {
      Guitar guitar = i.next();
      // Ignore serial number since that's unique
      // Ignore price since that's unique
      if (searchGuitar.getBuilder() != guitar.getBuilder())
        continue;
      String model = searchGuitar.getModel().toLowerCase();
      if ((model != null) && (!model.equals("")) &&
          (!model.equals(guitar.getModel().toLowerCase())))
        continue;
      if (searchGuitar.getType() != guitar.getType())
        continue;
      if (searchGuitar.getBackWood() != guitar.getBackWood())
        continue;
      if (searchGuitar.getTopWood() != guitar.getTopWood())
        continue;
      matchingGuitars.add(guitar);
    }
    return matchingGuitars.iterator();
  }
}
```

Something is still wrong

Rick is not happy

Why didn't the Gibsons show up?
Maybe there really were none in stock.
What did the customer enter?

Something's *still* wrong

Three steps to great software

1. Make sure the software does what the customer wants
2. Apply good object-oriented principles
3. Strive for a maintainable, reusable design

Does the software do what the customer wants?

What's the problem?

Things that might be wrong

1. Not all properties are relevant
2. There is no wildcard value
3. There’s no way to select more than one value for a property
4. …

You know, when a customer searches for a guitar, they're providing a set of characteristics they are looking for, not a specific guitar. Maybe that's the problem.

A GuitarSpec is not a Guitar

GuitarSpec

Guitar

We can use UML to show this

Incremental changes

```
public class GuitarSpec {

  private Builder builder;
  private String model;
  private Type type;
  private Wood backWood;
  private Wood topWood;

  public GuitarSpec(Builder builder, String model, Type type,
                    Wood backWood, Wood topWood) {
    this.builder = builder;
    this.model = model;
    this.type = type;
    this.backWood = backWood;
    this.topWood = topWood;
  }

  public Builder getBuilder() {
    return builder;
  }…
```

Incremental changes (2)

```
public class Guitar {

  private String serialNumber;
  private double price;
  GuitarSpec spec;

  public Guitar(String serialNumber, double price,
                Builder builder, String model, Type type,
                Wood backWood, Wood topWood) {
    this.serialNumber = serialNumber;
    this.price = price;
    this.spec = new GuitarSpec(builder, model, type, backWood,
topWood);
  }

…

  public GuitarSpec getSpec() {
    return spec;
  }
}
```

Three steps to great software

1. Make sure the software does what the customer wants
2. Apply good object-oriented principles
3. Strive for a maintainable, reusable design

What principles did we apply?

Are there any more that come to mind?

A word on increments & iterations

Iterative development

- Repeating the steps in a development process over and over
- Shorter iterations are better

Incremental development

- Add a bit at a time
- No big bang

A lot of customers are looking for 12-string guitars. Maybe we'd better add that to the guitars' characteristics we record.
According to the contract, that's not what you asked for.

We can add the number of strings. It’s not a problem.

What do you need to change?

1 _____

2 _____

3 _____

4 _____

What do you need to change?

1 GuitarSpec

2 Inventory

3

4

Changes to GuitarSpec

```
public class GuitarSpec {

  private Builder builder;
  private String model;
  private Type type;
  private int numStrings;
  private Wood backWood;
  private Wood topWood;

  public GuitarSpec(Builder builder, String model, Type type,
                    int numStrings, Wood backWood, Wood topWood) {
    this.builder = builder;
    this.model = model;
    this.type = type;
    this.numStrings = numStrings;
    this.backWood = backWood;
    this.topWood = topWood;
  }
…

  public int getNumStrings() {
    return numStrings;
  }
…
}
```

Changes to Inventory

```
public Iterator<Guitar> search(GuitarSpec searchSpec) {
    List<Guitar> matchingGuitars = new LinkedList<Guitar>();
    for (Iterator<Guitar> i = guitars.iterator(); i.hasNext(); ) {
      Guitar guitar = i.next();
      GuitarSpec spec = guitar.getSpec();
      if (searchSpec.getBuilder() != spec.getBuilder())
        continue;
      String model = searchSpec.getModel().toLowerCase();
      if ((model != null) && (!model.equals("")) &&
          (!model.equals(spec.getModel().toLowerCase())))
        continue;
      if (searchSpec.getType() != spec.getType())
        continue;
      if (searchSpec.getBackWood() != spec.getBackWood())
        continue;
      if (searchSpec.getTopWood() != spec.getTopWood())
        continue;
      if (searchSpec.getNumStrings() != spec.getNumStrings())
        continue;
      matchingGuitars.add(guitar);
    }
    return matchingGuitars.iterator();
  }
```

Cool. That’s just what I wanted. You’re the best. Next time you’ve got my business again.

Three steps to great software

1. Make sure the software does what the customer wants
2. Apply good object-oriented principles
3. Strive for a maintainable, reusable design

That wasn't so hard!

Can we do better?

Maybe we've allocated the responsibilities incorrectly. There's a pattern called *Information Expert* that might be appropriate here.

O-O design is all about assigning responsibilities to the objects in our system.

Information Expert

Assign responsibility to the class that has the essential information—the <u>information expert</u>.

Craig Larman, “Applying UML and Patterns”

Think about this?

What behavior is misplaced?

Who is the information expert?

What should we do?

Think about this?

What behavior is misplaced?

Matching the guitar to the specification

Who is the information expert?

GuitarSpec

What should we do?

Make GuitarSpec reasonable for determining

if it matches a guitar.

The new GuitarSpec class

```
public class GuitarSpec {
…
 public boolean matches(GuitarSpec otherSpec) {
    if (builder != otherSpec.builder)
      return false;
    if ((model != null) && (!model.equals("")) &&
        (!model.toLowerCase().equals(otherSpec.model.toLowerCase())))
      return false;
    if (type != otherSpec.type)
      return false;
    if (numStrings != otherSpec.numStrings)
      return false;
    if (backWood != otherSpec.backWood)
      return false;
    if (topWood != otherSpec.topWood)
      return false;
    return true;
  }
…
}
```

Gee, this looks sort of familiar!

The new Inventory class

```
Public class Inventory {
…
 public List search(GuitarSpec searchSpec) {
    List matchingGuitars = new LinkedList();
    for (Iterator i = guitars.iterator(); i.hasNext(); ) {
      Guitar guitar = (Guitar)i.next();
      if (guitar.getSpec().matches(searchSpec))
        matchingGuitars.add(guitar);
    }
    return matchingGuitars;
  }
…
}
```

Congratulations!

You've just performed your first refactoring

You've made the customer happy

You've applied solid O-O principles

You have a maintainable application

You've created great software, right?

Go out and celebrate at the coffee shop

How can you be sure?