Class Notes 2 (supplementary)

or

My Life with Shoes

or

An Introduction to Object Orientation using Java

Table of Contents

1. My Cool Shoes

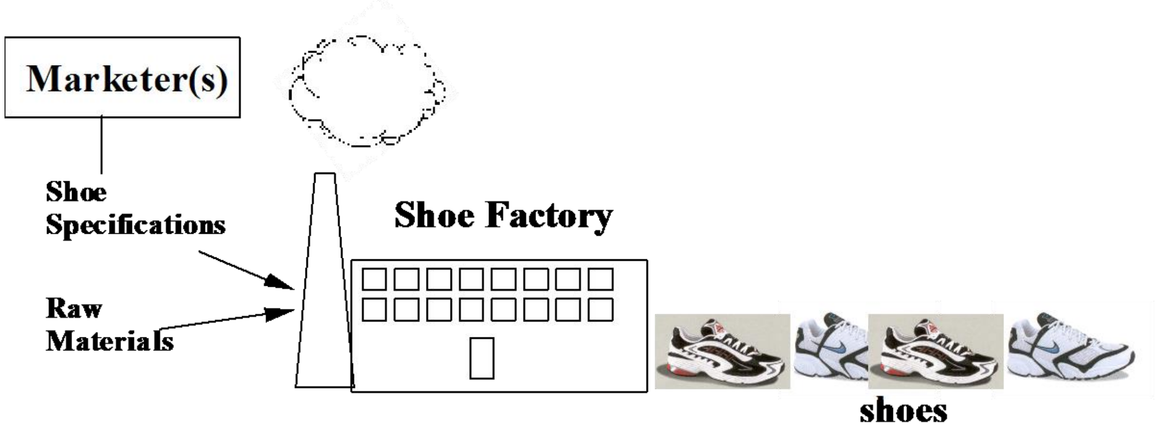
When I was growing up, my friends and I spent a lot of time worrying about shoes--the right shoes. Shoes that were in, shoes that had the right number of stripes or swooshes. Because--as you probably know--having the right shoes brought the admiration of friends, the respect of enemies and possibly even the attention of the opposite sex. We argued the merits of pneumatic foot attire vs. cantilevered Velcro accessories. Was it better to have shoes with laces that one did not tie or shoes with straps that one did not fasten? This was all very important stuff.

I've had a lot of experience with shoes, as I am sure you have as well, and that is why I would like to introduce some of the ideas behind Object Oriented Programming (OOP) from the point of view of footwear.

2. The Truth about Shoes

While the likes of Nike, Reebok and Adidas would have you believe that the shoes that they make are somehow better than their competitor's, shoes are basically very similar. I mean seriously--the idea is to get the shoe to break before your foot does.

By-and-large shoes are made in some third-world country from designs provided by the owner (marketer) of that particular shoe brand. For example, a factory in Indonesia might manufacture shoes for both Nike and Adidas using each company's respective designs—(adding labels at the end so you'll know who is gouging you). The plans for shoes, although often owned by different companies are pretty much the same--a shoe is a shoe is a shoe as it were.



3. Making Shoes

Millions of shoes are produced every year from the factories and those shoes are put on sale in pairs for us to buy. In fact, many things are put on sale. Go to your local mall and you're in consumer nirvana.

There are way more things on sale than any of us could ever buy by ourselves. I think Elvis once put it best when he said that, “if you nail any two objects together some schmuck will buy the new object off you”[1]. Actually, that is an interesting analogy, each one of these things on sale could be thought of as an ***object*** in the world, each different (we get really upset when someone hands us an apple when we're buying gum, for example), but each one having been made some place and having similarities to objects of the same type[2].

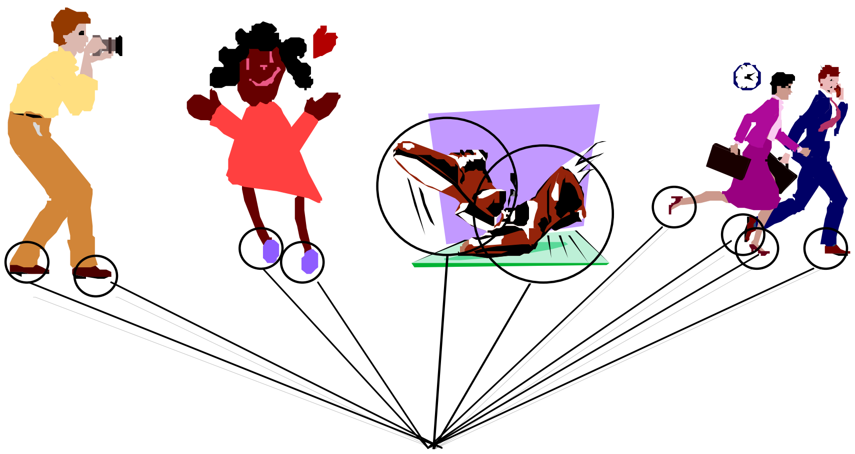
The idea of *similarity* is important. For example, your nose ring is an object but it's not the same as your shoe. I mean, they make nose rings in completely

[1] Actually, Elvis probably didn't say it but than again some people believe this guy is still alive and walking amongst us. Despite the fact that he would be a really old fat guy, it is conceivable that he may be reading this document. If this is, in fact the case, I would like to take this opportunity to apologize to Elvis for any damage I may have caused his reputation through this (probably erroneous) citation.
[2] Yes, I know, trees make apples not factories. However, you can think of a tree as an apple factory of sorts.

different factories than shoes[3]. Shoe companies don't usually make any money based on what you force into your nostril. And you do different things with your shoes than you do with your nose ring. Most people don't try and lace up their nose ring and also don't walk around on top of embedded rings on their feet—the point is, these objects are different *types* of objects. ***Think of "objects" as things in the world***.

All these shoe ***objects***, nose ring ***objects***, and all other ***objects*** belong to groups, these are groups of things that are very similar to…well, themselves. You might think of these things as belonging to a ***class*** of objects similar to themselves. ***Think of "classes" as the category (or type) of things that share properties like where they were made, how they behave and actions can reasonable be performed with them[4].***

In order to be part of a ***class***, ***objects*** need to have been created in similar ways.


Shoe Objects

4. Tying shoes

I learned to tie my shoes when it became embarrassing to have my mom do it. I took those laces and I found a ***method*** for putting them together so that they wouldn't drag on the ground and that could be undone quickly when mom wasn't around. My non-knuckle-dragging friends learned to do the same. Each one of our shoe ***objects*** had its own set of things you could do to it. For example, you could tie your right shoe and, unless something happened, that shoe would remain tied until you untied it. Although it doesn't sound like a big deal, if you tied your left shoe you didn't expect your right shoe to be tied automatically. Also, you didn't expect to tie your buddy's shoes by tying your own shoe.

[3] If you think about it, this is a good thing. Can you imagine the size of a Nike nose ring?
[4] Think about the nose ring vs. shoe analogy when you are trying to figure out what I mean.

In a nutshell, each shoe--mine, yours and everyone else's-- is an ***object***, is different from everyone else's and it has things you can do to it--an example being a ***method*** for tying it.

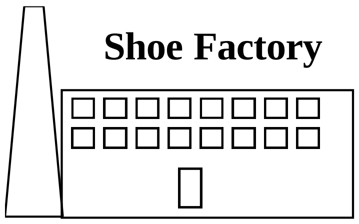
My Brown Loafers, All laces Tied.

So, let's make this a bit more like programming…or making shoes.

5. The Class -- the Shoe_factory

Lets start by making a factory that can manufacture shoes. This is the object-maker or factory or ***class***. Like any good factory you need a building or framework in which to put all the stuff used for manufacturing. This is done with the following code;

```
class Shoe_factory
{
            /* more stuff goes here */
}
```



Well, it's not much to look at.

You can think of this as the building that the shoe is going to be made in without anything in it. Now we need to add some basic facilities for making basic shoes. For simplicity, I will assume that this factory can only manufacture shoes with laces but the shoes it makes can be any colour.

6. Instance Variable -- Used to describe a shoe

When we describe the shoes we wear we normally name the characteristics of the shoe. For example I might refer to my "Pokemon yellow bath slippers"[5].

One of the things we need is a bunch of variables that will hold the values used to describe the shoes as they are produced. In OOP parlance, these are called ***instance variables***.

[5] Actually, these slippers are my daughter's…at least that is the story I'm sticking to!

```
          boolean tied = false;
          public String colour;
          public String brand;
```

The variable "*tied*" indicates if the shoe has been laced up. "*colour*" and "*brand*" are what they are named--they will hold the colour and brand of the shoe. These instance variables don't always need to have values supplied to them. In our case the values will be set when the shoe is actually made.

These instance variables get put in the factory like this. They can appear anywhere in the factory but people tend to put them first in the class.

```
     class Shoe_factory
     {
          boolean tied = false;
          public String colour;
          public String brand;

          /* more stuff goes here */
     }
```

7. Class Variables --keeping track of what the factory produces

The factory also wants to keep track of how many shoes it actually makes. We can do this with another form of variable. In this case the variable belongs to the factory and not the instance shoes. It is called a ***class variable***.

*You can tell a class variable from an instance variable because a class variable has the key word "**static**" in its declaration and an instance variable does not.*

```
     public static int shoes_made = 0;
```

We set this variable to zero since no shoes have been made yet--the factory is just getting started.

It is important to remember that this factory can make many shoes but there is only one factory. So, there is only one variable called "*shoes_made*" but there may be many variables with the name "*colour*" depending on how many shoes are made by the factory—each shoe will have its own *colour* instance variable.

1. This static (or class) variable gets put into the factory like this.

```
class Shoe_factory
{
        boolean tied = false;
        public String colour;
        public String brand;
        public static int shoes_made = 0;

        /* more stuff goes here
}
```

7. The Constructor -- Making a Shoe object

To make a shoe you have to ask the factory to "construct" it. To construct a shoe we need to start up the part of the factory that makes a shoe. In OOP this is called invoking the ***constructor***. In a sense, this is like pressing the “on” button of a machine.

The constructor for this class could look like this.

```
public Shoe_factory(String bra, String col)
{
        brand = bra;
        colour = col;
        shoes_made = shoes_made + 1;
        System.out.println
                ("Hurray, we made another shoe!");
}
```

The thing that is made by a constructor is called an ***object***. Note how the constructor has the same name as the class "Shoe_factory". This is always true.

In Java a *constructor* creates an *object*.

The two things inside the round braces are values being delivered to the constructor (these are called “arguments” or “parameters”)--kind of like Nike coming into a factory with a specific design for a shoe.

All the constructor does in this case is assign the formal values being supplied by the arguments to the instance variables that will hold them when the shoe is made. Our constructor also adds one to the number of shoes that have been made by this factory by updating the class variable "`shoes_made`".

Putting everything together we get.

```
class Shoe_factory
{
      boolean tied = false;
      public String colour;
      public String brand;
      public static int shoes_made = 0;

      public Shoe_factory(String bra, String col)
      {
            brand = bra;
            colour = col;
            shoes_made = shoes_made + 1;
            System.out.println
                  ("Hurray, we made another shoe!");
      }

      /* more stuff goes here
}
```

8. Instance Methods -- Shoes that Lace Up

Remember that I said that objects have certain actions that you can perform on them…like tying your shoelaces. These are called ***instance methods***. Each shoe will get a copy of its own instance methods as defined by the class--after all, it is a shoe (object) being made by the factory (class)—instance methods act on the object's own instance variables.

Instance methods perform actions on the data of the objects they are part of (instance variables).

I will add two methods that let you lace and unlace the shoe we constructed. Also remember what I said about tying and untying shoes--the method applies to a particular shoe (the object) not any shoe and certainly you can't "tie" the shoe factory.

```
public void lace(){
      tied = true;
      System.out.println("I am laced.");
}

public void unlace(){
      tied = false;
      System.out.println("I am unlaced.");
}
```

The code gets added to the Shoe_factory class but is really must be thought of as the code of the shoes to be made.

```
class Shoe_factory{
      boolean tied = false;
      public String colour;
      public String brand;
      public static int shoes_made = 0;

      public Shoe_factory(String bra, String col){
            brand = bra;
            colour = col;
            shoes_made = shoes_made + 1;
            System.out.println
                  ("Hurray, we made another shoe!");
      }

      public void lace(){
            tied = true;
            System.out.println("I am laced.");
      }

      public void unlace(){
            tied = false;
            System.out.println("I am unlaced.");
      }

      /* more stuff goes here
}
```

9. Another Instance Method

We can look at the values and state of individual shoe objects by creating an appropriate method to do so. The one below does this.

```
public void shoestatus(){
      System.out.print("My Brand: " + brand);
      System.out.print(", my Colour: " + colour);
      System.out.println(" laced?: " + tied);
      System.out.println();
}
```

Remember

You can think of the `shoestatus()` method as being analogous to looking down at your shoes and seeing what state they are in.

This method will simply dump all the values associated with a particular shoe. This method is simply added to the ones already there.

```
     class Shoe_factory{
          boolean tied = false;
          public String colour;
          public String brand;
          public static int shoes_made = 0;

          public Shoe_factory(String bra, String col)
          {
               brand = bra;
               colour = col;
               shoes_made = shoes_made + 1;
               System.out.println
                    ("Hurray, we made another shoe!");
          }

          public void lace(){
               tied = true;
               System.out.println("I am laced.");
          }

          public void unlace(){
               tied = false;
               System.out.println("I am unlaced.");
          }

          public void shoestatus(){
               System.out.print("My Brand: " + brand);
               System.out.print(", my Colour: " + colour);
               System.out.println(" laced?: " + tied);
               System.out.println();
          }

          /* more stuff goes here
}
```

10. Class Method--Finding Out About the Factory

It would be nice to be able to find out how many shoes that the factory actually made.

This has nothing to do with the individual shoes but is a function of the factory (or class). Because this value is part of the factory we say that the method used to provide the value is called a "***class method***". This is shown below.

```
     public static void how_many_shoes(){
          System.out.println("This factory has manufactured "
               + shoes_made + " shoe(s).");
          System.out.println();
     }
```

A class method has the ***static*** key word in its declaration. This indicates that this method will be a ***class method*** just like "`shoes_made`" had ***static*** in its declaration to make it a ***class variable***.

We use this method to output the count of shoes made by the factory. Our factory now looks like this.

```
      class Shoe_factory{
            boolean tied = false;
            public String colour;
            public String brand;
            public static int shoes_made = 0;
            public Shoe_factory(String bra, String col){
                  brand = bra;
                  colour = col;
                  shoes_made = shoes_made + 1;
                  System.out.println
                        ("Hurray, we made another shoe!");
            }

            public void lace(){
                  tied = true;
                  System.out.println("I am laced.");
            }

            public void unlace(){
                  tied = false;
                  System.out.println("I am unlaced.");
            }

            public void shoestatus(){
                  System.out.print("My Brand: " + brand);
                  System.out.print(", my Colour: " + colour);
                  System.out.println(" laced?: " + tied);
                  System.out.println();
            }

            public static void how_many_shoes(){
                  System.out.println
                        ("This factory has manufactured "
                        + shoes_made + " shoe(s).");
                  System.out.println();
            }
            /* more stuff goes here
}
```

Next we need to drive the actual process of making shoes. In our case I will use the main method (every Java program contains a main method…somewhere) to make the shoes. This could look like the following.

```
    public static void main(String[] args)
        {
            System.out.println
            ("Factory, how many shoes have you made?");
            how_many_shoes();


            Shoe_factory runner;
            System.out.println("Factory, make a Silver Nike.");
            runner = new Shoe_factory("Nike","Silver");

            System.out.println
            ("Factory, now how many shoes have you made?");
            how_many_shoes();

            System.out.println
            ("Factory, make a brown Buster brown shoe.");
            Shoe_factory casual =
                new Shoe_factory("Buster Brown", "Brown");

            System.out.println
            ("Factory, now how many shoes have you made?");
            how_many_shoes();

            System.out.println
            ("What is the status of each shoe?");
            runner.shoestatus();
            casual.shoestatus();

            System.out.println("Lace up my runner.");
            runner.lace();

            System.out.println
            ("What is the status of my shoes now?");
            runner.shoestatus();
            casual.shoestatus();
        }
```

Let's look at the first few line of the main method.

```
            Shoe_factory runner;
```

This tells the `Shoe_factory` class that you may want it to make a shoe (object) called "runner".

```
            runner = new Shoe_factory("Nike","Silver");
```

This line tells the factory to actually make the shoe called "runner" by invoking the ***constructor*** "`Shoe_factory`". The values "Nike" and "Silver" will be used by the constructor to set values in the shoe.

```
            Shoe_factory casual =
                new Shoe_factory("Buster Brown", "Brown");
```

In this line we do the two steps above in one line of code. Note that this is another shoe called "casual". It is not the same as the shoe called “runner”.

11. The complete Shoe_factory

Have a look at the code for the complete shoe factory.

```
class Shoe_factory{
      boolean tied = false;
      public String colour;
      public String brand;
      public static int shoes_made = 0;

      public void lace(){
            shoe_message();
            tied = true;
            System.out.println("I am laced.");
      }

      public void unlace(){
            shoe_message();
            tied = false;
            System.out.println("I am unlaced.");
      }

      public void shoestatus(){
            shoe_message();
            System.out.print("My Brand: " + brand);
            System.out.print(", my Colour: " + colour);
            System.out.println(" laced?: " + tied);
            System.out.println();
      }

      public Shoe_factory(String bra, String col){
            factory_message();
            brand = bra;
            colour = col;
            shoes_made = shoes_made + 1;
            System.out.println
                  ("Hurray, we made another shoe!");
      }

      public static void how_many_shoes(){
            factory_message();
            System.out.println
                  ("This factory has manufactured " +
                  shoes_made + " shoe(s).");
            System.out.println();
      }

      private void shoe_message(){
            /* Simple instance method to print out a message */
            System.out.print("A message from a shoe: ");
      }

      private static void factory_message(){
            /* Simple class method to print out a message */
            System.out.print("A message from the factory: ");
      }
}
```

12. Engaging the factory to Make Shoes -- A Driver Class

Going with the discussion I provided above. The "factory" we have created in the ***class*** called `Shoe_factory` can be used to make ***objects*** (shoes) of that class. For example we can create another class that can use the `Shoe_factory` class to make shoes. In the example below, the class `Shoe_factory_driver` has a main function that uses the methods of the `Shoe_factory` class. This is kind of like Nike or Adidas sending their specifications to the Shoe factory and having those specs turned into shoes specific to them. The driver class is shown below.

```
class Shoe_factory_driver{
      public static void main(String[] args){
            System.out.println
            ("Factory, how many shoes have you made?");
            Shoe_factory.how_many_shoes();

            Shoe_factory runner;
            System.out.println
            ("Factory, make a Silver Nike.");
            runner = new Shoe_factory("Nike","Silver");

            System.out.println
            ("Factory, now how many shoes have you made?");
            Shoe_factory.how_many_shoes();

            System.out.println
            ("Factory, make a brown Buster brown shoe.");
            Shoe_factory casual =
                  new Shoe_factory("Buster Brown", "Brown");

            System.out.println
            ("Factory, now how many shoes have you made?");
            Shoe_factory.how_many_shoes();

            System.out.println
            ("What is the status of each shoe?");
            runner.shoestatus();
            casual.shoestatus();

            System.out.println("Lace up my runner.");
            runner.lace();

            System.out.println
            ("What is the status of my shoes now?");
            runner.shoestatus();
            casual.shoestatus();
      }

}
```

13. A sample conversation between the factory, yourself and shoes.

When you compile and run the `Shoe_factory` class and the `Shoe_factory_driver` class the output will be similar to that below. Try and follow through the program above and see how the output is produced.

```
Factory, how many shoes have you made?
A message from the factory: This factory has manufactured 0 shoe(s).

Factory, make a Silver Nike.
A message from the factory: Hurray, we made another shoe!
Factory, now how many shoes have you made?
A message from the factory: This factory has manufactured 1 shoe(s).

Factory, make a brown Buster brown shoe.
A message from the factory: Hurray, we made another shoe!
Factory, now how many shoes have you made?
A message from the factory: This factory has manufactured 2 shoe(s).

What is the status of each shoe?
A message from a shoe: My Brand: Nike, my Colour: Silver laced?: false

A message from a shoe: My Brand: Buster Brown, my Colour: Brown laced?: false

Lace up my runner.
A message from a shoe: I am laced.
What is the status of my shoes now?
A message from a shoe: My Brand: Nike, my Colour: Silver laced?: true

A message from a shoe: My Brand: Buster Brown, my Colour: Brown laced?: false
```

14. An Odd Ending

The preceding has been a simplified discussion of some of the more important features of OOP for the beginning programmer. OOP is a very powerful and a rather simple concept once you "get it". Try making your own version of the `Shoe_factory` using whatever product you like (`Nose_ring_maker` might be a good place to start).

Recall that this is all about me. While I was trying to select the “in” shoe to actually purchase I noticed that I was without sufficient funds to purchase anything but Bata North Stars. This turned out to work out just fine as I was able to start a conversation with my future wife who had the same predicament when we met.