**Core Java**

**Difference between Procedural and Object oriented programming**

**A Real-World Example**

Okay, that's enough theory. We're going to put both types of programming to the test with a real-world example. Let's say that you are working for a vehicle parts manufacturer that needs to update it's online inventory system. Your boss tells you to program two similar but separate forms for a website, one form that processes information about cars and one that does the same for trucks.

For cars, we will need to record the following information:

* Color
* Engine Size
* Transmission Type
* Number of doors

For trucks, the information will be similar, but slightly different. We need:

* Color
* Engine Size
* Transmission Type
* Cab Size
* Towing Capacity

In procedural programming, you would write the code first to process the car form and then the code for the truck form.

With object-oriented programming, you would write a base class called vehicle that would record the common characteristics what we need from both trucks and cars. In this case, the vehicle class will record:

* Color
* Engine Size
* Transmission Type

We'll make each one of those characteristics into a separate method. The color method, for example, could take the color of the vehicle as a parameter and do something with it, like storing it in a database.

Next, we will create two more classes: truck and car, both of which will inherit all of the methods of the vehicle class and extend it with methods that are unique to them.

The car class will have a method called numberOfDoors and the truck class will have the methods cabSize and towingCapacity.

Okay, so let's assume that we have a working example for both procedural and OO programming. Now, let's run through a few scenarios that we could come across in a normal working environment. You know the type of scenario because it always begins with the thought: I really wish my boss didn't send this in an email request at 4pm on a Friday afternoon.

**Scenario 1**

Suppose that we suddenly need to add a bus form, that records the following information:

* Color
* Engine Size
* Transmission Type
* Number of passengers

*Procedural:* We need to recreate the entire form, repeating the code for Color, Engine Size, and Transmission Type.

*OOP:* We simply extend the vehicle class with a bus class and add the method, numberOfPassengers.

**Scenario 2**

Instead of storing color in a database like we previously did, for some strange reason our client wants the color emailed to him.

*Procedural:* We change three different forms: cars, trucks, and buses to email the color to the client rather than storing it in the database.

*OOP:* We change the color method in the vehicle class and because the car, truck, and bus classes all extend (or inherit from, to put it another way) the vehicle class, they are automatically updated.

**Scenario 3**

We want to move from a generic car to specific makes, for example: Nissan and Mazda.

*Procedural:* We create a new form for each make, repeating all of the code for generic car information and adding the code specific to each make.

*OOP:* We extend the car class with a nissan class and a mazda class and add methods for each set of unique information for that car make.

**Scenario 4**

We found a bug in the transmission type area of our form and need to fix it.

*Procedural:* We open and update each form.

*OOP:* We fix the transmissionType method in the vehicle class and the change perpetuates in every class that inherits from it.