Práctica 3
Abstract Factory

Undav

Diseño de Sistemas

Alumno: Brian Sosa

Profesor: Martín Machuca

Consignas

1) Se requiere modelar un programa haciendo uso del patrón Abstract Factory.

Necesitamos fabricar las piezas de distintos vehículos para luego ensamblarlos y obtener un producto final.

Las diferentes partes son:

- Rueda
- Puerta
- Capot
- Chasis
- Cuadro
- Volante
- Manubrio
- Asiento
- Motor
- Caño de escape
- Cadena

Con esas piezas se desea construir los siguientes vehículos:

- Motocicleta
- Automóvil
- Bicicleta

Deseamos tener distintas fábricas de objetos que provean las piezas específicas para cada tipo de vehículo.

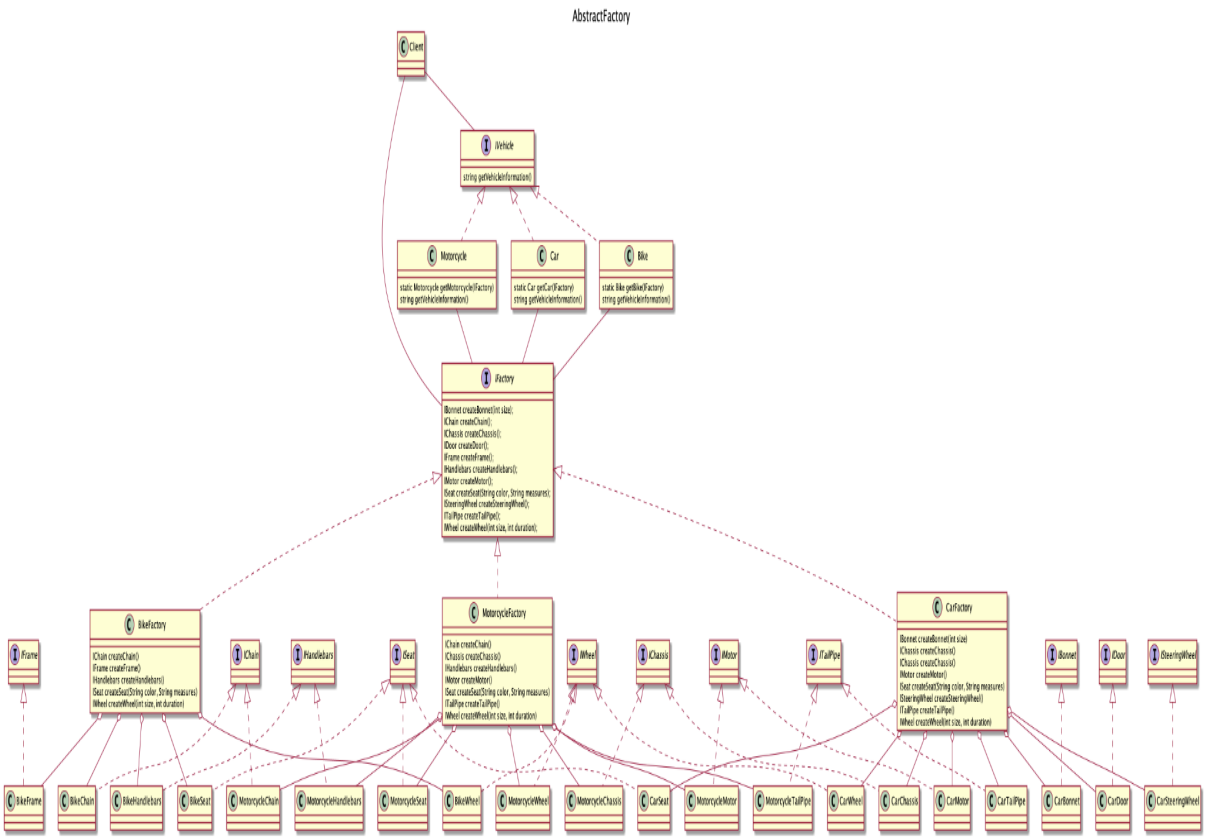
Implementarlo usando un lenguaje de programación orientado a objetos.

2) Realizar un diagrama de clases.

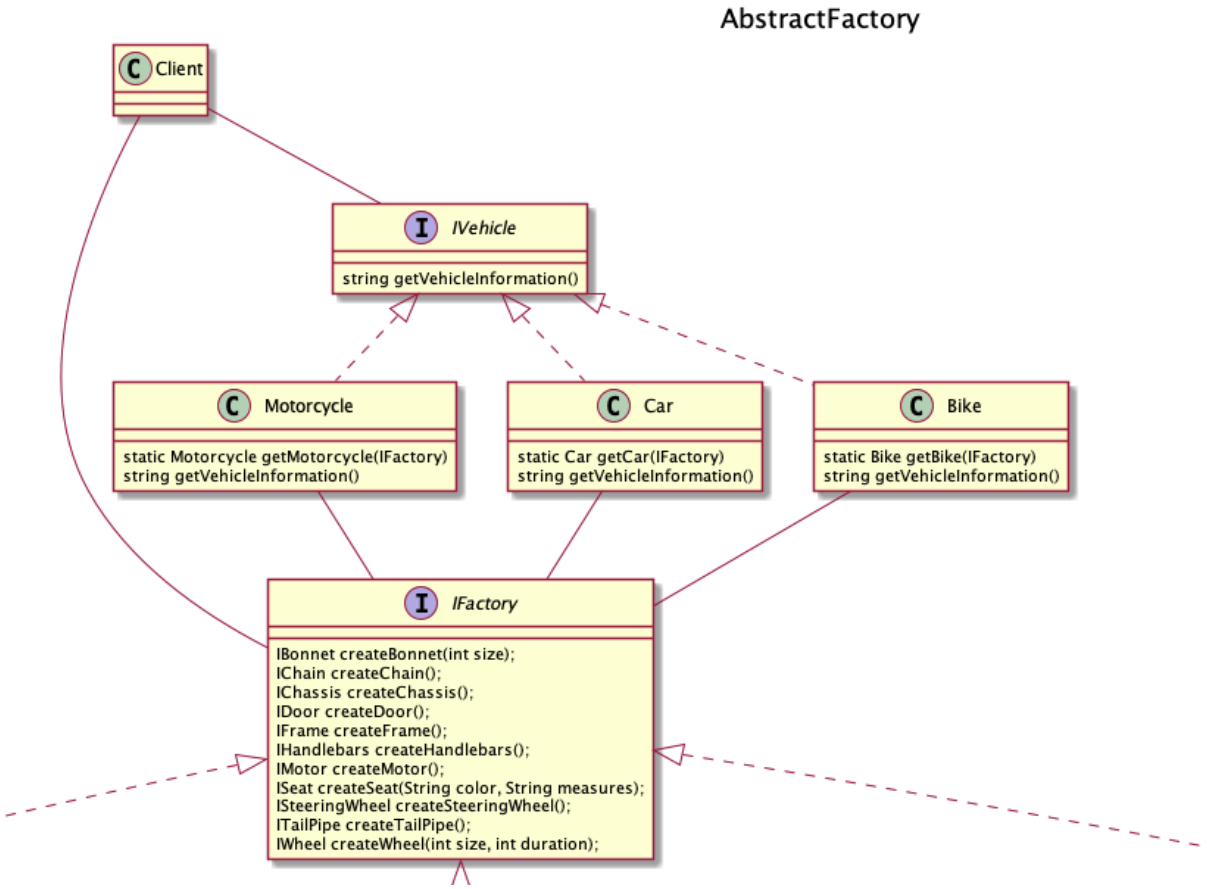
Realización

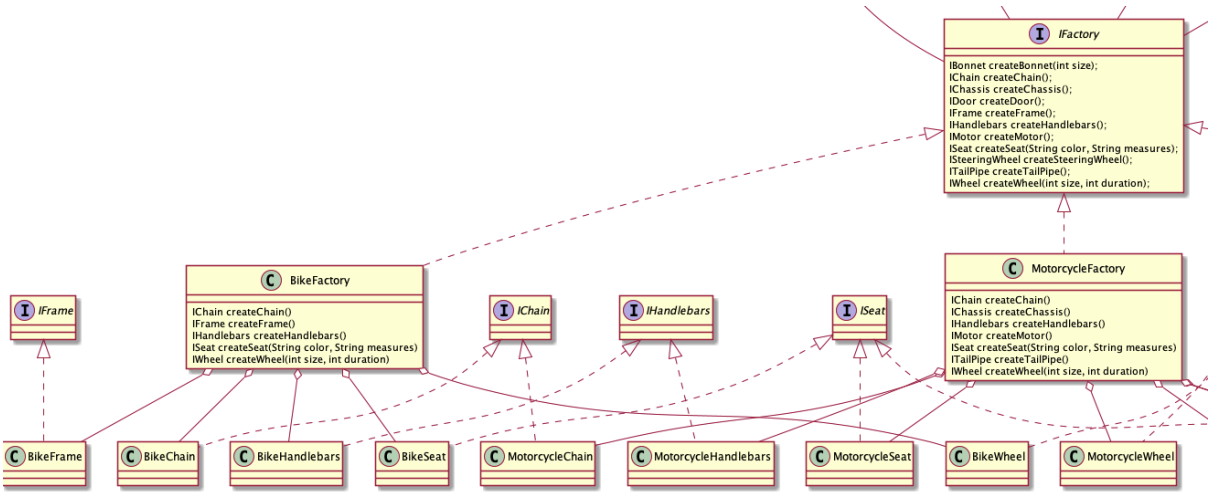
Diagrama de clases

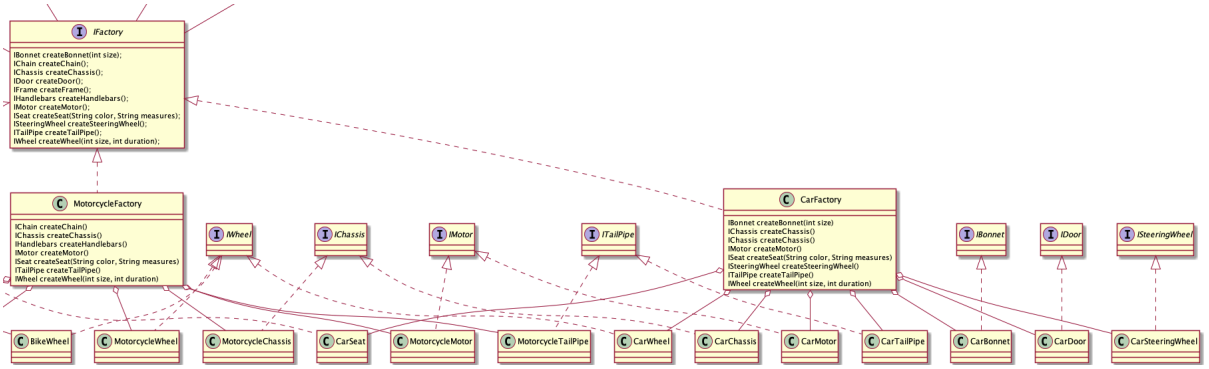
A continuación se deja el diagrama de clases para tener una visión general del programa que se va a explicar en el próximo punto.
Diagrama completo:



Haciendo un poco de zoom:







Realización del programa

El programa se planteó a modo de crear las 3 fábricas requeridas (auto, moto y bici) y obtener información referente al modelo creado, cada uno con sus características.

Para la explicación del programa primero se realiza la explicación de los productos, para ir de un nivel más bajo a abstracción hasta el más alto que sería el programa cliente.

Aclaración: para simplificar la explicación y evitar la repetición de código solo se va a explicar los productos Wheel y Seat

Productos

Cada producto tiene una interfaz que refiere a los métodos que debe cumplir cada producto.

Interfaces

```
package AbstractFactory.Products.Wheel;

public interface IWheel {
```

```
   String getInformation();
}
```

```
package AbstractFactory.Products.Seat;

public interface ISeat {
   String getInformation();
}
```

Luego, tenemos las implementaciones concretas de cada interfaz creando un producto que refiere a un vehículo en particular.

Clases concretas que implementan IWheel

```
package AbstractFactory.Products.Wheel;

public class MotorcycleWheel implements IWheel {
   private int size;
   private int duration;

   public MotorcycleWheel(int size, int duration){
       this.size = size;
       this.duration = duration;
   }

   public String getInformation() {
       return String.format("La rueda de la moto es rodado %d y tiene una
duracion de %d años", this.size, this.duration);
   }
}
```

```
package AbstractFactory.Products.Wheel;

public class CarWheel implements IWheel {
   private int size;
   private int duration;

   public CarWheel(int size, int duration){
       this.size = size;
       this.duration = duration;
   }

   public String getInformation() {
       return String.format("La rueda del auto es rodado %d y tiene una
duracion de %d años", this.size, this.duration);
   }
}
```

```
package AbstractFactory.Products.Wheel;

public class BikeWheel implements IWheel {
   private int size;

   public BikeWheel(int size){
       this.size = size;
   }
```

```
    public String getInformation() {
        return String.format("La rueda de la bici es rodado %d", this.size);
    }
}
```

Clases concretas que implementan ISeat

```
package AbstractFactory.Products.Seat;

public class BikeSeat implements ISeat {
    private String measures;

    public BikeSeat(String measures){
        this.measures = measures;
    }

    public String getInformation() {
        return String.format("El asiento de la bici es acolchada y de medidas
%s", this.measures);
    }
}
```

```
package AbstractFactory.Products.Seat;

public class CarSeat implements ISeat {
    private String color;

    public CarSeat(String color){
        this.color = color;
    }

    public String getInformation() {
        return String.format("El asiento del auto es una butaca de color %s",
this.color);
    }
}
```

```
package AbstractFactory.Products.Seat;

public class MotorcycleSeat implements ISeat {
    private String color;
    private String measures;

    public MotorcycleSeat(String color, String measures){
        this.color = color;
        this.measures = measures;
    }

    public String getInformation() {
        return String.format("El asiento de la moto es antideslizante de color
%s y medidas %s", this.color, this.measures);
    }
}
```

Con esto logramos abstraer el método “getInformation” de cada producto concreto a nivel de interfaz.

Fábricas

Las fábricas tienen una interfaz en común que comprenden a los métodos creacionales de cada producto.

IFactory

```
package AbstractFactory.Factories;

public interface IFactory {
   IBonnet createBonnet(int size);
   IChain createChain();
   IChassis createChassis();
   IDoor createDoor();
   IFrame createFrame();
   IHandlebars createHandlebars();
   IMotor createMotor();
   ISeat createSeat(String color, String measures);
   ISteeringWheel createSteeringWheel();
   ITailPipe createTailPipe();
   IWheel createWheel(int size, int duration);
}
```

La interfaz se implementa en cada una de las 3 fábricas. Cada una de estas crea el producto concreto que corresponde al modelo que se quiere crear.
Cabe aclarar que al estar utilizando la misma interfaz, no hay una correcta segregación de interfaces y hay métodos que devuelven null ya que hay productos que no corresponden a la fábrica que se quiere crear. Pero se omite esta falencia para poder concentrarnos en la correcta estructura del patrón.

BikeFactory

```
package AbstractFactory.Factories;

public class BikeFactory implements IFactory {
   public IBonnet createBonnet(int size) {
       return null;
   }

   public IChain createChain() {
       return new BikeChain();
   }

   public IChassis createChassis() {
       return null;
   }

   public IDoor createDoor() {
       return null;
   }

   public IFrame createFrame() {
       return new BikeFrame();
   }
```

```
    public IHandlebars createHandlebars() {
        return new BikeHandlebars();
    }

    public IMotor createMotor() {
        return null;
    }

    public ISeat createSeat(String color, String measures) {
        return new BikeSeat(measures);
    }

    public ISteeringWheel createSteeringWheel() {
        return null;
    }

    public ITailPipe createTailPipe() {
        return null;
    }

    public IWheel createWheel(int size, int duration) {
        return new BikeWheel(size);
    }
}
```

CarFactory

```
package AbstractFactory.Factories;


public class CarFactory implements IFactory {
    public IBonnet createBonnet(int size) {
        return new CarBonnet(size);
    }

    public IChain createChain() {
        return null;
    }

    public IChassis createChassis() {
        return new CarChassis();
    }

    public IDoor createDoor() {
        return new CarDoor();
    }

    public IFrame createFrame() {
        return null;
    }

    public IHandlebars createHandlebars() {
        return null;
    }

    public IMotor createMotor() {
        return new CarMotor();
    }

    public ISeat createSeat(String color, String measures) {
```

```
        return new CarSeat(color);
    }

    public ISteeringWheel createSteeringWheel() {
        return new CarSteeringWheel();
    }

    public ITailPipe createTailPipe() {
        return new CarTailPipe();
    }

    public IWheel createWheel(int size, int duration) {
        return new CarWheel(size, duration);
    }
}
```

MotorcycleFactory

```
package AbstractFactory.Factories;

public class MotorcycleFactory implements IFactory {
    public IBonnet createBonnet(int size) {
        return null;
    }

    public IChain createChain() {
        return new MotorcycleChain();
    }

    public IChassis createChassis() {
        return new MotorcycleChassis();
    }

    public IDoor createDoor() {
        return null;
    }

    public IFrame createFrame() {
        return null;
    }

    public IHandlebars createHandlebars() {
        return new MotorcycleHandlebars();
    }

    public IMotor createMotor() {
        return new MotorcycleMotor();
    }

    public ISeat createSeat(String color, String measures) {
        return new MotorcycleSeat(color, measures);
    }

    public ISteeringWheel createSteeringWheel() {
        return null;
    }

    public ITailPipe createTailPipe() {
        return new MotorcycleTailPipe();
    }
```

```
    public IWheel createWheel(int size, int duration) {
        return new MotorcycleWheel(size, duration);
    }
}
```

Modelos

Con modelos me refiero a los objetos del dominio: Auto, Moto y Bici. Los modelos implementan una interfaz “IVehicle” que contiene un método para obtener información sobre ese objeto.

IVehicle

```
package AbstractFactory.Models;

public interface IVehicle {
    String getVehicleInformation();
}
```

Luego para los modelos, reciben en el constructor una fábrica con la cual obtienen los productos para realizar el ensamblado de partes y obtener el producto final.

Bike

```
package AbstractFactory.Models;

public class Bike implements IVehicle {
    private IWheel[] wheels = new IWheel[2];
    private ISeat seat;
    private IFrame frame;
    private IChain chain;
    private IHandlebars handlebars;

    public Bike(IFactory factory){
        this.wheels[0] = factory.createWheel(14, 3);
        this.wheels[1] = factory.createWheel(14, 3);
        this.seat = factory.createSeat("negro", "30x25");
        this.frame = factory.createFrame();
        this.chain = factory.createChain();
        this.handlebars = factory.createHandlebars();
    }

    public String getVehicleInformation(){
        return String.format("" +
            "La bici cuenta con %d ruedas caracterizadas por: %s. %n" +
            "%s. %n" +
            "%s. %n" +
            "%s. %n" +
            "%s.%n",
            this.wheels.length, this.wheels[0].getInformation(),
            this.seat.getInformation(),
            this.frame.getInformation(),
            this.chain.getInformation(),
            this.handlebars.getInformation());
    }
}
```

Car

```
package AbstractFactory.Models;

public class Car implements IVehicle{
   private IWheel[] wheels = new IWheel[4];
   private IDoor[] doors = new IDoor[4];
   private IBonnet bonnet;
   private IChassis chassis;
   private IMotor motor;
   private ISeat[] seats = new ISeat[3];
   private ISteeringWheel steeringWheel;
   private ITailPipe tailPipe;

   public Car(IFactory factory){
       this.wheels[0] = factory.createWheel(17, 3);
       this.wheels[1] = factory.createWheel(17, 3);
       this.wheels[2] = factory.createWheel(17, 3);
       this.wheels[3] = factory.createWheel(17, 3);
       this.doors[0] = factory.createDoor();
       this.doors[1] = factory.createDoor();
       this.doors[2] = factory.createDoor();
       this.doors[3] = factory.createDoor();
       this.bonnet = factory.createBonnet(160);
       this.chassis = factory.createChassis();
       this.motor = factory.createMotor();
       this.seats[0] = factory.createSeat("negro", "70x90");
       this.seats[1] = factory.createSeat("negro", "70x90");
       this.seats[2] = factory.createSeat("negro", "180x90");
       this.steeringWheel = factory.createSteeringWheel();
       this.tailPipe = factory.createTailPipe();
   }

   public String getVehicleInformation(){
       return String.format("" +
           "El auto cuenta con %d ruedas caracterizadas por: %s. %n" +
           "Es un auto de %d puertas caracterizadas por: %s. %n" +
           "Contiene %d asientos. %s. %n" +
           "%s. %n" +
           "%s. %n" +
           "%s. %n" +
           "%s. %n" +
           "%s.%n",
           this.wheels.length, this.wheels[0].getInformation(),
           this.doors.length, this.doors[0].getInformation(),
           this.seats.length, this.seats[0].getInformation(),
           this.bonnet.getInformation(),
           this.chassis.getInformation(),
           this.motor.getInformation(),
           this.steeringWheel.getInformation(),
           this.tailPipe.getInformation());
   }
}
```

Motorcycle

```
package AbstractFactory.Models;

import AbstractFactory.Factories.IFactory;
```

```
public class Motorcycle implements IVehicle {
   private IWheel[] wheels = new IWheel[2];
   private IChassis chassis;
   private IMotor motor;
   private ISeat seat;
   private ITailPipe tailPipe;
   private IChain chain;
   private IHandlebars handlebars;

   public Motorcycle(IFactory factory){
       this.wheels[0] = factory.createWheel(14, 3);
       this.wheels[1] = factory.createWheel(14, 3);
       this.chassis = factory.createChassis();
       this.motor = factory.createMotor();
       this.seat = factory.createSeat("negro", "30x25");
       this.tailPipe = factory.createTailPipe();
       this.chain = factory.createChain();
       this.handlebars = factory.createHandlebars();
   }

   public String getVehicleInformation(){
       return String.format("" +
           "La moto cuenta con %d ruedas caracterizadas por: %s. %n" +
           "%s. %n" +
           "Motor %s. %n" +
           "%s. %n" +
           "%s. %n" +
           "%s. %n" +
           "%s.%n",
           this.wheels.length, this.wheels[0].getInformation(),
           this.chassis.getInformation(),
           this.motor.getInformation(),
           this.seat.getInformation(),
           this.tailPipe.getInformation(),
           this.chain.getInformation(),
           this.handlebars.getInformation());
   }
}
```

Cliente

El código cliente interactúa solo con las fábricas concretas y con los modelos concretos, donde se abstrae toda la creación de los productos dentro de las fábricas y cada una con sus características específicas.
En este código cliente se trata de obtener la información de cada uno de los modelos (Car, Motorcycle y Bike) que a su vez utilizan la información que posee el producto concreto que creó cada fábrica.

Clase Cliente

```
package AbstractFactory;

public class Client {
   public static void main(String[] args) {
       IFactory carFactory = new CarFactory();
       IFactory motorcycleFactory = new MotorcycleFactory();
       IFactory BikeFactory = new BikeFactory();
```

```
        IVehicle[] vehicles = new IVehicle[]{new Car(carFactory), new
Motorcycle(motorcycleFactory), new Bike(BikeFactory)};

        for (int i = 0; i < vehicles.length; i++){
            System.out.println("INFO VEHICLE");
            System.out.println(vehicles[i].getVehicleInformation());
        }
    }
}
```

Output terminal

```
INFO VEHICLE
El auto cuenta con 4 ruedas caracterizadas por: La rueda del auto es rodado
17 y tiene una duracion de 3 años.
Es un auto de 4 puertas caracterizadas por: puerta blindada.
Contiene 3 asientos. El asiento del auto es una butaca de color negro.
El capot del auto tiene una medida de 160.
Chasis de auto super copado.
Motor v8 4.0 TFSI 420CV 550Nm.
Volante antideslizante con luces que se prenden al ritmo de la música.
Caño de escape que te escuchan hasta Gerli cuando cruzas el Pueyrredon.

INFO VEHICLE
La moto cuenta con 2 ruedas caracterizadas por: La rueda de la moto es rodado
14 y tiene una duracion de 3 años.
Chasis de moto irrompible.
Motor Nose de motores de moto.
El asiento de la moto es antideslizante de color negro y medidas 30x25.
El mejor caño escape para hacer cortes con la moto.
Esta es la mejor cadena de moto.
manubrio de moto.

INFO VEHICLE
La bici cuenta con 2 ruedas caracterizadas por: La rueda de la bici es rodado
14.
El asiento de la bici es acolchada y de medidas 30x25.
Cuadro de bici que si te caes no se dobla.
Esto es una buena cadena de bici.
manubrio de bici.
```