LP00

Trabajo

- Ensayo N° 06

Nombre del profesor:

- Hancco Carpio, Rony Jordan

Nombre del alumno:

- Negrete Girano, Luis Carlo

Lima, 11 de junio del 2015

SEXTO ENSAYO SOBRE LENGUAJE DE PROGRAMACIÓN ORIENTADOS A OBJETOS

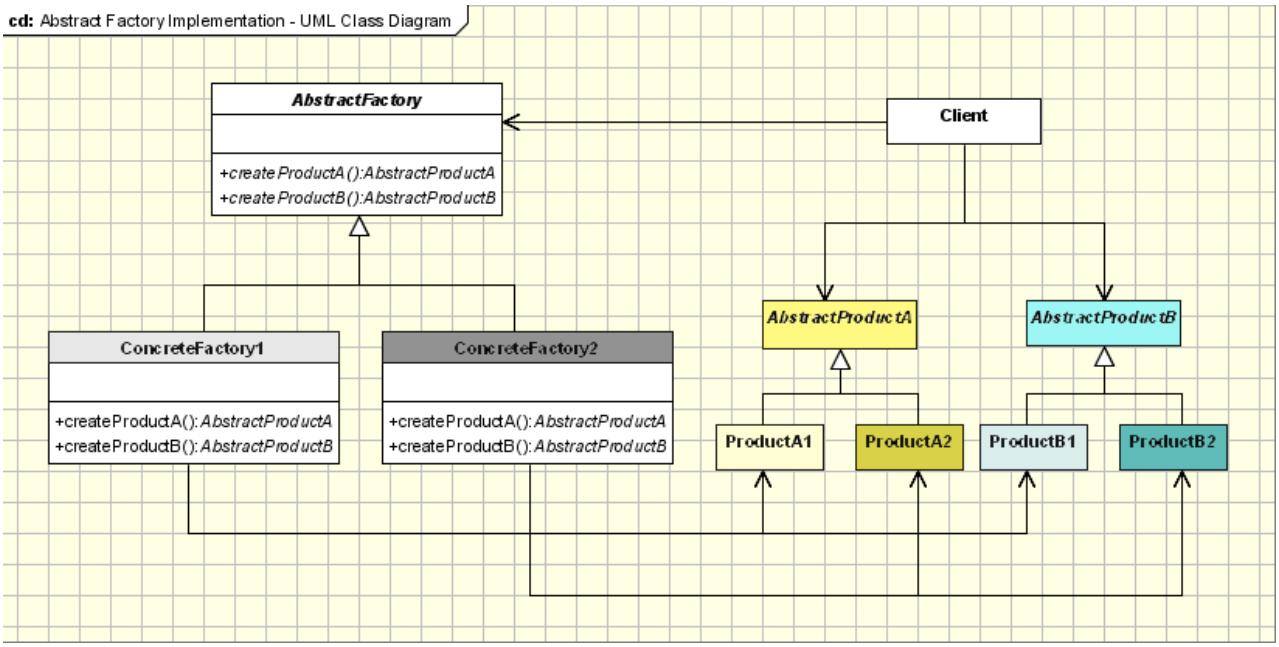
Este sexto ensayo, en particular, tendrá un solo tema, el cual trata acerca del patrón "*Abstract Factory".*

En primer lugar, debemos precisar que el "*Abstract Factory"* quiere decir en nuestro idioma "factoría abstracta", y este se trata de un patrón de diseño. Ahora bien, qué es un patrón de diseño, este es parte de la estructura fundamental de las soluciones a problemas comunes en el desarrollo de software." Es decir, nos proporcionan una solución ya probada a problemas de desarrollo de software que están en contextos símiles. Cabe mencionar que los elementos de un patrón de diseño son los siguientes: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios). Así también, debemos rescatar que esto trabaja únicamente con interfaces.

Otro tópico a revisar es el concepto de "Factory", el cual es un objeto que surge para la creación de otros objetos, en otras palabras, un *Factory* es un objeto que simplemente retorna un objeto de algún método. Asimismo, debemos revisar lo que significa el *patrón Factory* el cual es un patrón creacional que utiliza métodos de los *Factory* para poder solucionar problemas relacionados a la creación de objetos sin necesidad de especificar la clase exacta del objeto que va a ser creado. Podría decirse que estaría emulando la función del constructor, pero de una manera distinta.

Una vez asimilado este concepto, podemos pasar a entender lo que hace la "*abstract factory*" pues esta nos permite encapsular, de alguna manera, un grupo de *factories* individuales que tienen algo en común (por lo general un interfaz) sin necesidad de especificar las clases concretas. Nótese que una clase abstracta, es una clase de la cual no se pueden definir instancias u objetos; mientras que, una clase concreta es lo opuesto.

A continuación, pasaremos a ver cómo se implementa y cómo trabaja el *Abstract Factory*, a través del siguiente diagrama UML.



Aquí, podemos apreciar, que las clases participantes en el patrón *Abstract Factory* son:

Abstract Factory, el cual declara un interfaz de operaciones que crea productos abstractos.

Abstract Product, el cual declara una interfaz para un tipo de objetos productos.

Product, define un producto que será creado por el *Concrete Factory* correspondiente. Asimismo, implementa la interfaz *Abstract Product.*

Concrete Factory, implementa operaciones para crear productos concretos.

Client, es quien usa el interfaz creado por las clases del *Abstract Factory* y el *Abstract Product.*

Por último, pasaremos a lo que concierne cómo trabajar con el *Abstract Factory* en el IDE, en primer lugar debemos declararlo de la siguiente manera:

```
public interface Algo {
    public ServicioAlgo crearAlgo();
}
```

Posteriormente, los métodos que serán empleados en los productos concretos, aparecerán en los productos abstractos, tal y como se muestra.

```
public interface ServicioAlgo {
    public void HacerAlgo1();
    public void HacerAlgo2();
    public void HacerAlgo3();
}
```

Ahora, los productos concretos creados, tienen la particularidad de que la clase creada ahora usa el "implements" para referirse a los métodos declarados en los productos abstractos, podría decirse que es algo simétrico a lo que se usaba en herencia cuando se utilizaba el "extends".

```
public class Servicio1 implements ServicioAlgo {

    @Override
    public void HacerAlgo1() {
        System.out.println("Ahora");
    }

    @Override
    public void HacerAlgo2() {
        System.out.println("No");
    }

    @Override
    public void HacerAlgo3() {
        System.out.println("Joven");
    }
}
```

```
public class Servicio2 implements ServicioAlgo {

    @Override
    public void HacerAlgo1() {
        System.out.println("Más");
    }

    @Override
    public void HacerAlgo2() {
        System.out.println("Tarde");
    }

    @Override
    public void HacerAlgo3() {
        System.out.println("Joven");
    }
}
```

Para finalizar con el ensayo, hablaremos de los *Concrete Factory*, los cuales se encargan de implementar a la *Abstract Factory*, lo que hace es instanciar el producto concreta que le corresponde, tal y como se muestra.

```
public class Cosa1Factory implements ServicioAlgo{

    @Override
    public ServicioAlgo crearServicio() {
        return new ServicioCosa1();
    }

}
```