Abstract Factory

Nombre:
Abstract Factory

Clasificación del patrón:
Creacional

Intención:
Provee una interface para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas

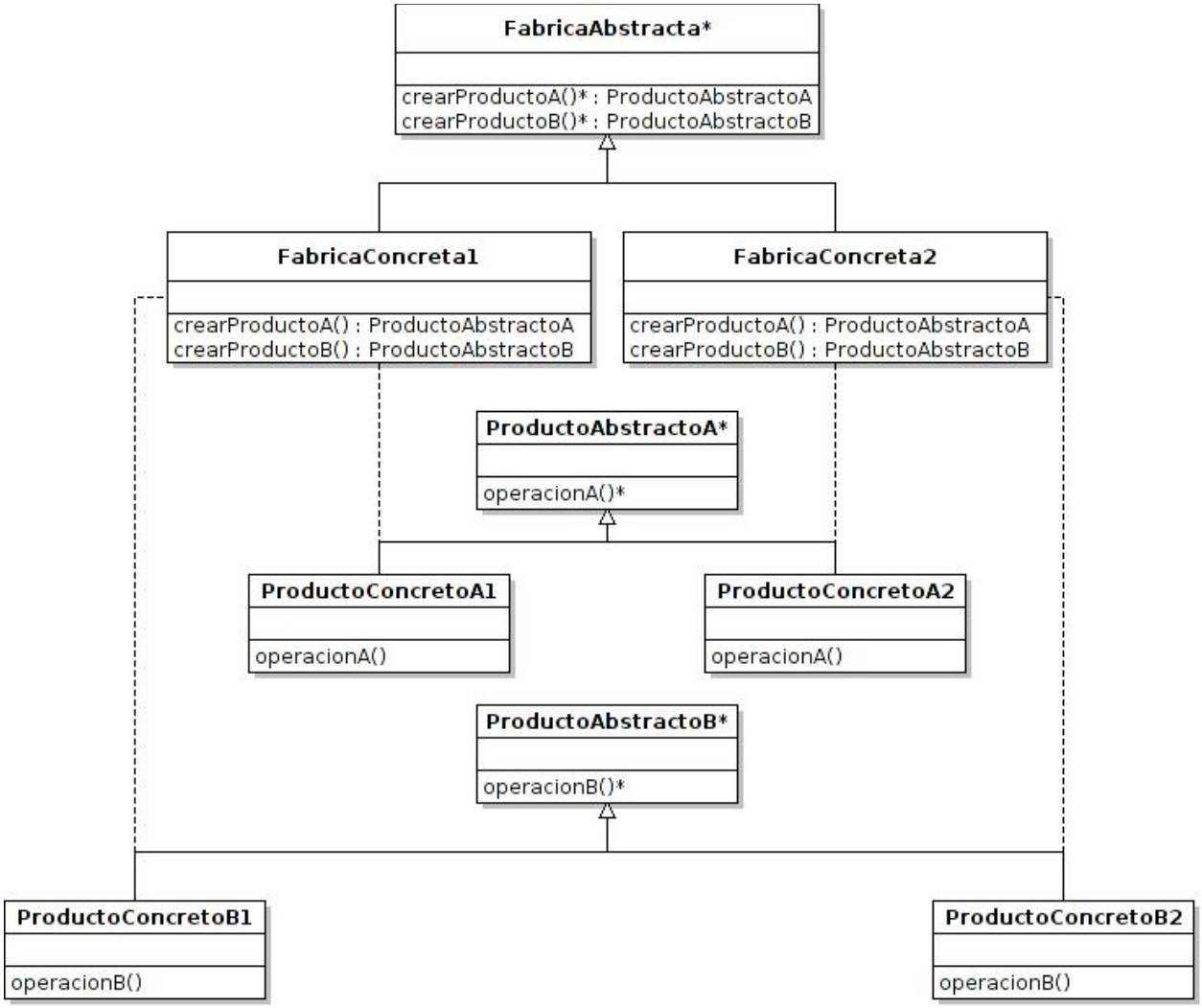
Otros nombres:
Kit, Fábrica Abstracta

Motivación:
Teniendo un conjunto de herramientas que soportan varias representaciones. El patrón de diseño soluciona el problema de que el cliente deba cambiar si cambia la interfaz de usuario, convirtiendo lo anterior en una falacia. Con la implementación se pueden obtener varios look and feel reduciendo el consumo de recursos del programa y facilitando su flexibilidad.

Aplicabilidad:
Se debe usar cuando:

- Una aplicación no debe depender de cómo son creadas las instancias de la clase.
- El servicio debe ser configurado con una de múltiples familias de clases.
- El software debe usar clases solamente de una familia a la vez.

Estructura:



Participantes:

- Cliente: Es el que quiere obtener la instancia de alguno de los productos, llamará a la factoría adecuada.
- Factoría Abstracta: Debe proporcionar un método para obtener cada objeto que pueda ser creado.
- Fábricas Concretas: Son las familias de productos, envían la instancia concreta de la familia que van a crear.
- Producto Abstracto: Definición de interfaces para las familias de productos genéricos.
- Producto Concreto: Implementación de los diferentes productos.

Colaboraciones:
Generalmente la instancia Fábrica Concreta se crea en tiempo de ejecución, ésta crea el objeto producto con una implementación en particular. Para crear diferentes Productos los clientes deberán usar diferentes Fábricas Concretas.

Ventajas:

- Contribuye a la elaboración de un diseño ligeramente acoplado.
- Restringe el uso de clases de una familia o una configuración de familias a la vez.
- El soporte a nuevas configuraciones es fácil.
- Promueve la consistencia entre clases.

Desventajas:

- El número de clases que pueden ser instanciados por cada familia es definido en la interface del Abstract Factory.
    - Hace difícil añadir nuevas clases.

Implementación:

- Fábricas como Singleton: Usualmente solo se requiere una instancia de cada Factoría Concreta.
- Creando productos: Fábrica Abstracta declara una interfaz para la creación de productos, en las clases de Productos Concretos es donde realmente se crean ellos mismos.
- Definición de factorías extensibles: Factoría Abstracta define operaciones por cada producto. La aparición de productos hace obligatoria la modificación de la interfaz Factoría Abstracta.

Código de ejemplo:

- Clase reloj:

```
1  public abstract class Reloj {
2
3      abstract String dameLaHora();
4  }
```

- Formato AM_PM:

```
public class RelojAmPm extends Reloj{

    public RelojAmPm(){

    }

    public String dameLaHora() {
        Date d = new Date();
        int hora = d.getHours();
        int minutos = d.getMinutes();
        int segundos = d.getSeconds();
        String tr;
        if (hora&lt;=12){
            tr="Son las "+hora+":"+minutos+":"+segundos+" AM";
        } else {
            tr="Son las "+(hora-12)+":"+minutos+":"+segundos+" PM";
        }

        return tr;
    }

}
```

- Formato 24 horas:

```
public class Reloj24Hrs extends Reloj {

    public String dameLaHora() {
        Date d = new Date();
        int hora = d.getHours();
        int minutos = d.getMinutes();
        int segundos = d.getSeconds();
        String tr;
        tr = "Son las " + hora + ":" + minutos + ":" + segundos + " ";

        return tr;
    }
}
```

- Manejador de instancias:

```
1  public class RelojFactory {
2      public static final int RELOJ_AM_PM=0;
3      public static final int RELOJ_24_HRS=1;
4
5      public RelojFactory(){
6
7      }
8
9      public static Reloj createReloj(int tipoDeReloj){
10         if (tipoDeReloj==RelojFactory.RELOJ_24_HRS){
11             return new Reloj24Hrs();
12         }
13         if (tipoDeReloj==RelojFactory.RELOJ_AM_PM){
14             return new RelojAmPm();
15         }
16
17         return null;
18     }
19
20 }
```

- Clase cliente:

```
1  public class MainClient {
2
3      public static void main(String[] args) {
4          Reloj r = RelojFactory.createReloj(RelojFactory.RELOJ_24_HRS);
5          System.out.println(r.dameLaHora());
6      }
7  }
```

Usos conocidos:

- ET++ para archivar portablemente sobre diferentes sistemas windows.

Patrones relacionados:

- Prototype
- Singleton
- Factory Method

Bibliografía:

No específico. (No específico). GoF Design Patterns (Versión 2.1.0) [Aplicación móvil]. Descargado de: https://drive.google.com/file/d/0BywiVyFlIabXcVhGZlJBcnhWTkU/view.

Abstract Factory [Página web]. (10 de octubre de 2018). Ubicación: https://www.ecured.cu/Abstract_Factory.

Zuluaga, P. (21 de abril de 2015). Abstract Factory. Patrones de Software. http://patronesdedis.blogspot.com/2015/04/abstract-factory.html.

Junta de Andalucía. (s.f). Factoría Abstracta. Marco de Desarrollo de la Junta de Andalucía. http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/191.