Laboratorio 16

Patrones de Diseño: Abstract Factory

1. Competencias

1.1. Competencias del curso

Conoce, comprende e implementa programas con el lenguaje de programación C++ siguiendo la correcta utilización de los diferentes patrones de diseño.

1.2. Competencia del laboratorio

Conoce, comprende e implementa programas usando el lenguaje de programación C++, utilizando el patrón Abstract Factory

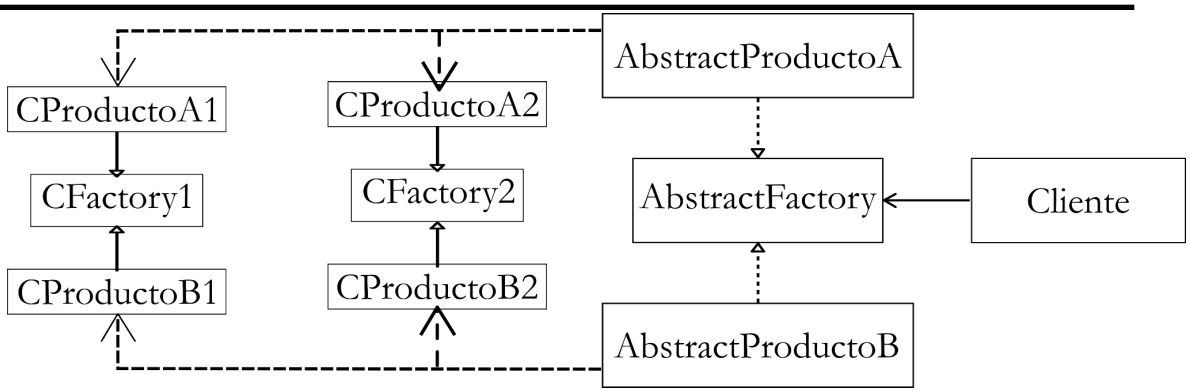
2. Equipos y Materiales

- Un computador.
- IDE para C++.
- Compilador para C++.

3. Marco Teórico

3.1. Introducción

El patrón de diseño Abstract Factory permite producir objetos con una relación en común, pero sin una especificación concreta de las clases. El problema a solucionar ocurre cuando se tienen varios tipos posibles de un producto y además existen varias variaciones entre ellos. Para esto se desea crear un objeto que contenga un tipo de un producto y una variante específica. Esto suele presentarse cuando se escriben componentes de software que son multiplataforma, como los elementos de una interfaz gráfica. Bajo este escenario se pueden crear objetos tipo Botones que pueden tener formatos diferentes dependiendo del sistema operativo, pero ambos cumplirán la misma tarea.



En la imagen anterior se muestra el modelo estándar del patrón Abstract Factory. La clase *AbstractFactory* mantiene la lista de productos que pueden ser implementados por el *Cliente* de forma genérica.

```
class AbstractFactory {
public:
      virtual AbstractProductoA* CrearProductoA() const = 0;
      virtual AbstractProductoB* CrearProductoB() const = 0;
};
```

La clase *AbstractProductoA* contiene la interfaz a utilizar por los componentes de productos A, y del cual pueden surgir una familia de productos.

```
class AbstractProductoA {
public:
       virtual ~AbstractProductoA() {};
       virtual std::string Funcion1_A() const = 0;
};
```

Las siguientes clases (*CProductoA1* y CProductoA*2*) son derivadas de la clase *AbstractProductoA*.

```
class CProductoA1 : public AbstractProductoA {
public:
       std::string Funcion1_A() const override {
              return "Producto A1.";
       }
};

class CProductoA2 : public AbstractProductoA {
       std::string Funcion1_A() const override {
              return "Producto A2.";
```

```
        }
};
```

La clase *AbstractProductoB* al igual que la clase *AbstractProductoA*, contiene la interfaz a utilizar por los componentes de productos B, y del cual pueden surgir una familia de productos.

```
class AbstractProductoB {
public:
        virtual ~AbstractProductoB() {};
        virtual std::string Funcion1_B() const = 0;
        virtual std::string Funcion2_B(const AbstractProductoA& colaborador) const = 0;
};
```

Lista de productos que implementan a *AbstractProductoB*

```
class CProductoB1 : public AbstractProductoB {
public:
        std::string Funcion1_B() const override {
                return "Producto B1.";
        }
        std::string  Funcion2_B(const  AbstractProductoA&  colaborador)  const
override {
                const std::string result = colaborador.Funcion1_A();
                return "B1 con ayuda de " + result;
        }
};
class CProductoB2 : public AbstractProductoB {
public:
        std::string Funcion1_B() const override {
                return "Producto B2.";
        }
        std::string  Funcion2_B(const  AbstractProductoA&  colaborador)  const
override {
                const std::string result = colaborador.Funcion1_A();
                return "B2 con ayuda de " + result;
        }
};
```

Las clases *CFactory1* y *CFactory2*, son los productos finales que son producidos utilizando los productos *AbstractProductoB* y AbstractProductoA con características diferentes.

```
class CFactory1 : public AbstractFactory {
public:
        AbstractProductoA* CrearProductoA() const override {
```

```
                return new CProductoA1();
        }
        AbstractProductoB* CrearProductoB() const override {
                return new CProductoB1();
        }
};

class CFactory2 : public AbstractFactory {
public:
        AbstractProductoA* CrearProductoA() const override {
                return new CProductoA2();
        }
        AbstractProductoB* CrearProductoB() const override {
                return new CProductoB2();
        }
};
```

Las clase del Cliente, se encarga de producir objetos en combinación de los productos A y Productos B. Simplemente se le indica qué tipos de objeto *AbstractFactory* será utilizado para generar el producto deseado.

```
void Cliente(const AbstractFactory& f) {
        const AbstractProductoA* producto_a = f.CrearProductoA();

        const AbstractProductoB* producto_b = f.CrearProductoB();
        std::cout << producto_b->Funcion1_B() << "\n";
        std::cout << producto_b->Funcion2_B(*producto_a) << std::endl;
        delete producto_a;
        delete producto_b;
}

int main() {
        std::cout << "Cliente: Tipo 1 ";
        CFactory1* f1 = new CFactory1();
        Cliente(*f1);
        delete f1;
        std::cout << std::endl;
        std::cout << "Cliente: Tipo 2 ";
        CFactory2* f2 = new CFactory2();
        Cliente(*f2);
        delete f2;
        return 0;
}
```

El resultado de la ejecución es:

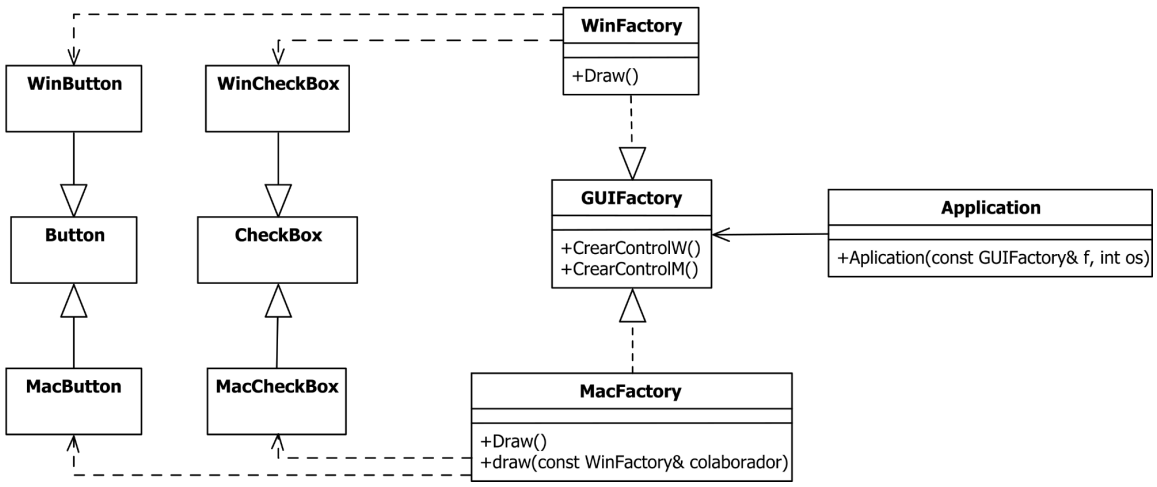
```
Cliente: Tipo 1 Producto B1
B1 con ayuda de Producto A1

Cliente: Tipo 2 Producto B2
B2 con ayuda de Producto A2
```

1. Ejercicios

Resolver los siguientes ejercicios planteados:

1. Dado el siguiente modelo de la siguiente imagen, realizar la implementación del modelo. De ser posible, incluir una interfaz para Linux que también sea utilizado por los productos Button y CheckBox. (Las funciones Draw() solo imprimen el tipo de Producto y el sistema en que se encuentra)



La función main debe ser:

```
int main() {
	std::cout << "Cliente: Windows ";
	Button* f1 = new Button();
	Aplication(*f1, 1); // 1 - Windows
	delete f1;
	std::cout << std::endl;

	std::cout << "Cliente: Mac ";
	Button* f2 = new Button();
```

```
        Aplication(*f2, 2); // 2 - Mac
        delete f2;
        return 0;
}
```

La salida espera debe ser:

Cliente : Windows
Dibujando Button Windows

Cliente: Mac
Dibujando Button Mac

2. Entregables

Al final estudiante deberá:

1. Compactar el código elaborado y subirlo al aula virtual de trabajo. Agregue sus datos personales como comentario en cada archivo de código elaborado.
2. Elaborar un documento que incluya tanto el código como capturas de pantalla de la ejecución del programa. Este documento debe de estar en formato PDF.
3. El nombre del archivo (comprimido como el documento PDF), será su LAB13_GRUPO_A/B/C_CUI_1erNOMBRE_1erAPELLIDO.

   (Ejemplo: LAB13_GRUPO_A _2022123_PEDRO_VASQUEZ).
4. Debe remitir el documento ejecutable con el siguiente formato:

   LAB13_GRUPO_A/B/C_CUI_ EJECUTABLE_1erNOMBRE_1erAPELLIDO

   (Ejemplo: LAB13_GRUPO_A_EJECUTABLE_2022123_PEDRO_VASQUEZ).

En caso de encontrarse trabajos similares, los alumnos involucrados no tendrán evaluación y serán sujetos a sanción.