UNIVERSIDAD DE LIMA | 60 años

DISEÑO DE SOFTWARE CON PATRONES

UNIDAD 2: DISEÑO E IMPLEMENTACIÓN

Temario

- Patrón de Diseño en Ingeniería del Software.
- Categoría de patrones.
- Patrones creacionales: Factory Method, Abstract Factory, Singleton.

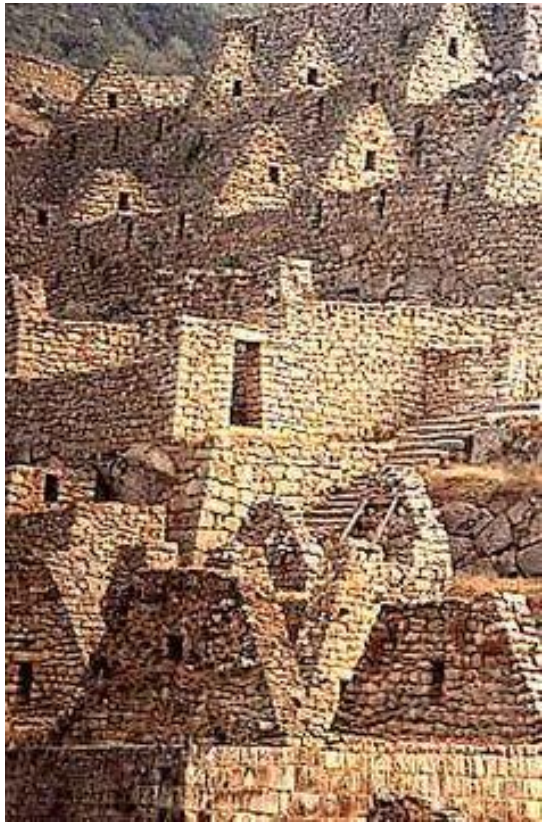
UNIVERSIDAD DE LIMA | 60 años

Patrón de Diseño en Ingeniería del Software

Los patrones y los lenguajes de patrones son formas de describir las mejores prácticas, los buenos diseños y capturar la experiencia de manera que sea posible que otros reutilicen esta experiencia

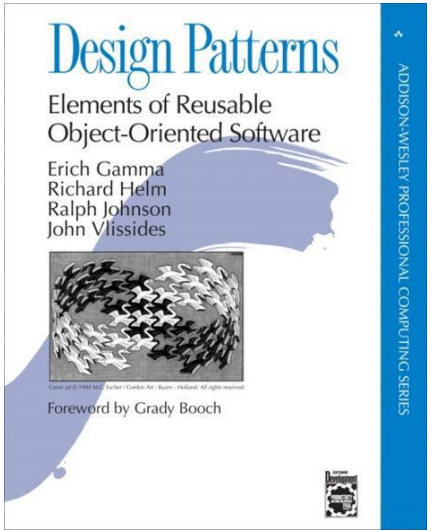
PATRONES DE DISEÑO

- 1977, Christopher Alexander, Arquitecto
  - "Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar esa solución un millón de veces, sin hacer lo mismo dos veces".
- 1994 – "Gang of Four" (GOF)
  - Reconocido libro "Design Patterns: Elements of Reusable Object-Oriented Software"



PATRONES DE DISEÑO

- 1994 – “Gang of Four” (GOF)
  - Reconocido libro “Design Patterns: Elements of Reusable Object-Oriented Software”
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides



UNIVERSIDAD DE LIMA

Partes de un Patrón

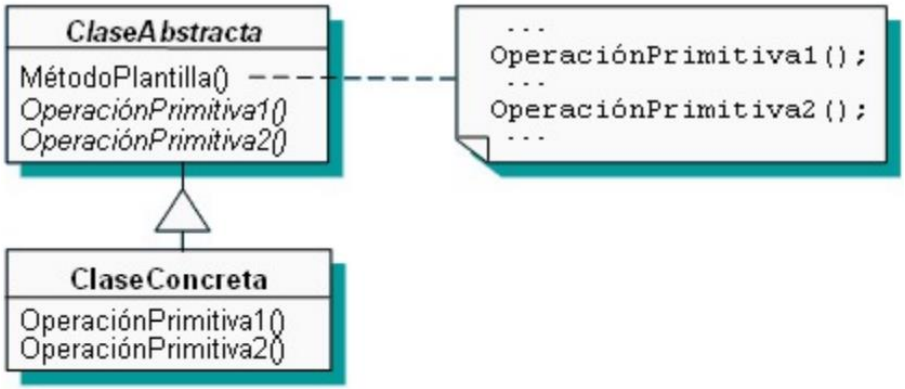
- **Nombre del patrón.-** Es un identificador que podemos usar para describir un problema de diseño.
- **El problema.-** Describe cuándo aplicar el patrón. Explica el problema y su contexto.
- **La solución.-** Describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboraciones.
- **Las consecuencias.-** son los resultados y las compensaciones de aplicar el patrón.

Partes de un Patrón

UNIVERSIDAD DE LIMA

Patrón de Diseño en Ingeniería del Software

- Soluciones reusables a problemas recurrentes que encontramos durante el desarrollo de software Orientado a Objetos y que han tenido éxito.
- Descripciones de clases y objetos relacionados que resolverán un problema de diseño general en un determinado contexto.



UNIVERSIDAD DE LIMA

UNIVERSIDAD DE LIMA | 60 años

Categoría de patrones

Hay muchos patrones de diseño, necesitamos una forma de organizar y para ello se utilizan los catálogos o categorías de patrones

Categorías en los patrones

A. **Por propósito** ¿Qué hace el patrón?
   1. **De Creación**: Establecen cómo deben crearse los objetos y clases. Ayudan a hacer el sistema independiente de cómo se crean, se componen y se representan sus objetos.
   2. **Estructurales**: Establecen cómo se combinan las clases y los objetos para formar estructuras más grandes.
   3. **De Comportamiento**: Se relaciona en modo en que las clases y objetos interactúan (algoritmo) y se reparten las responsabilidades.

Categorías en los patrones

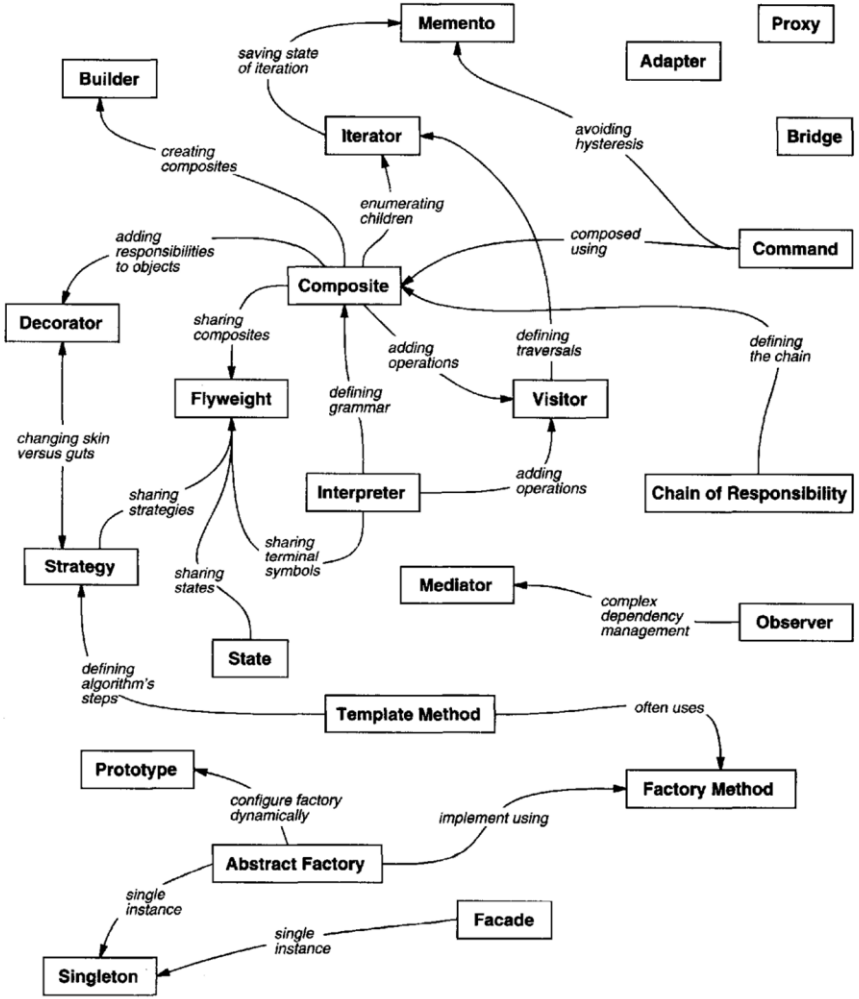
B. **Por ámbito**: ¿A que se aplica principalmente?

1. **Clases:** Relaciones estáticas entre clases y subclases (Herencia).
2. **Objetos:** Relaciones dinámicas entre objetos

Categorías en los patrones

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method (107)	Adapter (class) (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (object) (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Flyweight (195) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Observer (293) State (305) Strategy (315) Visitor (331)

Relación entre patrones



UNIVERSIDAD DE LIMA

¿Cómo describir un patrón?

UNIVERSIDAD DE LIMA

¿Cómo los patrones resuelven los problemas de diseño?

- Encontrar los objetos apropiados
- Determinar la granularidad de los objetos
- Especificar las interfaces
- Especificar las implementaciones

ENCONTRAR LOS OBJETOS APROPIADOS

- Un diagrama de clases de análisis modela el mundo del problema
- Un diagrama de clases de diseño modela la solución al problema.
- Un diagrama de clases de diseño incluye clases que vienen del análisis, pero hay otras clases que no tienen representación en el mundo real.
- **Los patrones de diseño ayudan a identificar abstracciones menos obvias.**

DETERMINAR LA GRANULARIDAD DE LOS OBJETOS

- Objetos varían de tamaño y número.
- Pueden representar desde pequeñas hasta grandes aplicaciones
- Los patrones de diseño ayudan a determinar la granularidad correcta de un diseño OO.
  - FALTA DE GRANULARIDAD: Pocas clases con muchas funcionalidades en cada una
  - EXCESO DE GRANULARIDAD: Muchas clases con funcionalidades reducidas

ESPECIFICAR LAS INTERFACES

- **Signatura de Operación:** Nombre de la operación, Objetos que serán parámetros y el valor de retorno.
- **Interfaz:** Conjunto de todas las signaturas de las operaciones de un objeto, es decir peticiones que se pueden enviar al objeto para realizar alguna operación definida en él.
- Una interfaz no implementa las operaciones, son los objetos quienes implementarán dichas operaciones.
- **Los patrones de diseño ayudan a definir las interfaces identificando sus elementos clave y los tipos de datos que se envían a la interface.**
- También especifican relaciones entre interfaces.

ESPECIFICAR LAS IMPLEMENTACIONES

- La implementación de un objeto queda definida por su clase.
- Una clase específica datos, la representación interna del objeto y define las operaciones que puede realizar.
- Los objetos se crean instanciando una clase.
- Uso de Herencia de Clases: Clase padre y Clases hijas
- Uso de Clase Abstracta

Causas comunes de rediseño

- Hay que tener en cuenta cómo puede necesitar cambiar el sistema a lo largo del tiempo.
- Los patrones de diseño ayudan a asegurar que un sistema pueda cambiar de formas concretas.
- Fuerte acoplamiento
- Añadir funcionalidad mediante la herencia
- Dependencia del HW o SW

¿Cómo seleccionar un patrón?

- Revisar la sección de propósito de cada patrón.
- Estudiar las otras secciones
- Revisar la relación entre patrones de diseño.
- Revisar patrones de propósito similar.
- Revisar si es necesario el rediseño. ¡No siempre aplicar un patrón es la solución, sino, todo lo contrario!

¿Cómo usar un patrón?

1. Leer el patrón de principio a fin para tener una perspectiva.
2. Estudiar las otras secciones
3. Revisar el código ejemplo
4. Elegir nombres significativos de acuerdo al contexto y patrón.
5. Definir las clases, declarar interfaces, herencia, y modificar las clases existentes para aplicar el patrón.
6. Implementar las operaciones para llevar a cabo las responsabilidades y colaboraciones del patrón.

UNIVERSIDAD DE LIMA | 60 años

Patrón Creacional: Factory Method

Establecen cómo deben crearse los objetos y clases

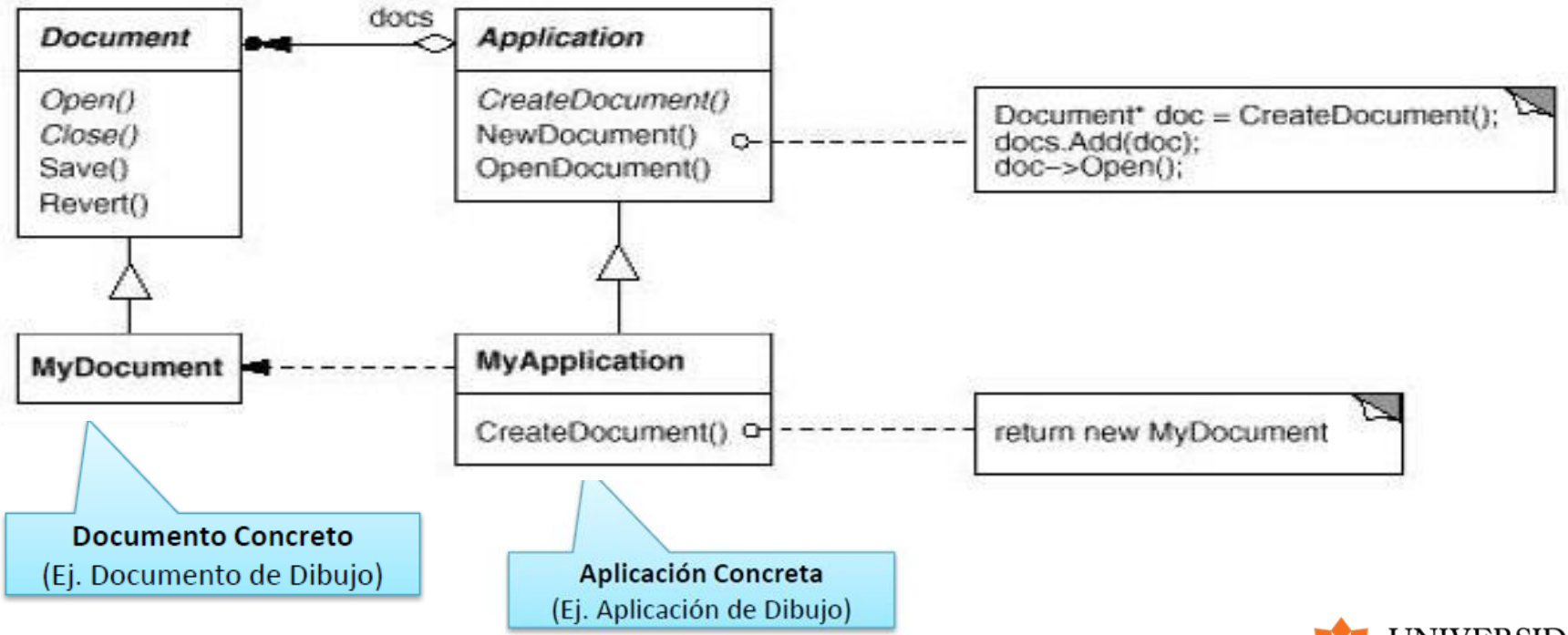
Patrón Creacional: Factory Method

- **PROPÓSITO**
  - Define una interfaz para crear un objeto pero deja que las subclases decidan qué clase instanciar.
  - Delega la función de crear objetos a las subclases.

Patrón Creacional: Factory Method

- **MOTIVACIÓN**
  - Los frameworks son estructuras definidas que utilizan clases abstractas para definir y mantener relaciones entre objetos y son responsables de crearlos.
  - Por ejemplo: En un framework de aplicaciones
    - **Clases abstractas:** Aplicación y Documento
    - **Clase concreta AplicaciónX**: La aplicación concreta a crear, que será una subclase de Aplicación(Una aplicación de Dibujo, de estadísticas). Existirá una clase concreta por cada tipo de aplicación que deseo crear.
    - **Clase concreta Documento X**: El documento concreto por el tipo de aplicación a crear (DocumentoDibujo)

Patrón Creacional: Factory Method



UNIVERSIDAD DE LIMA

Patrón Creacional: Factory Method

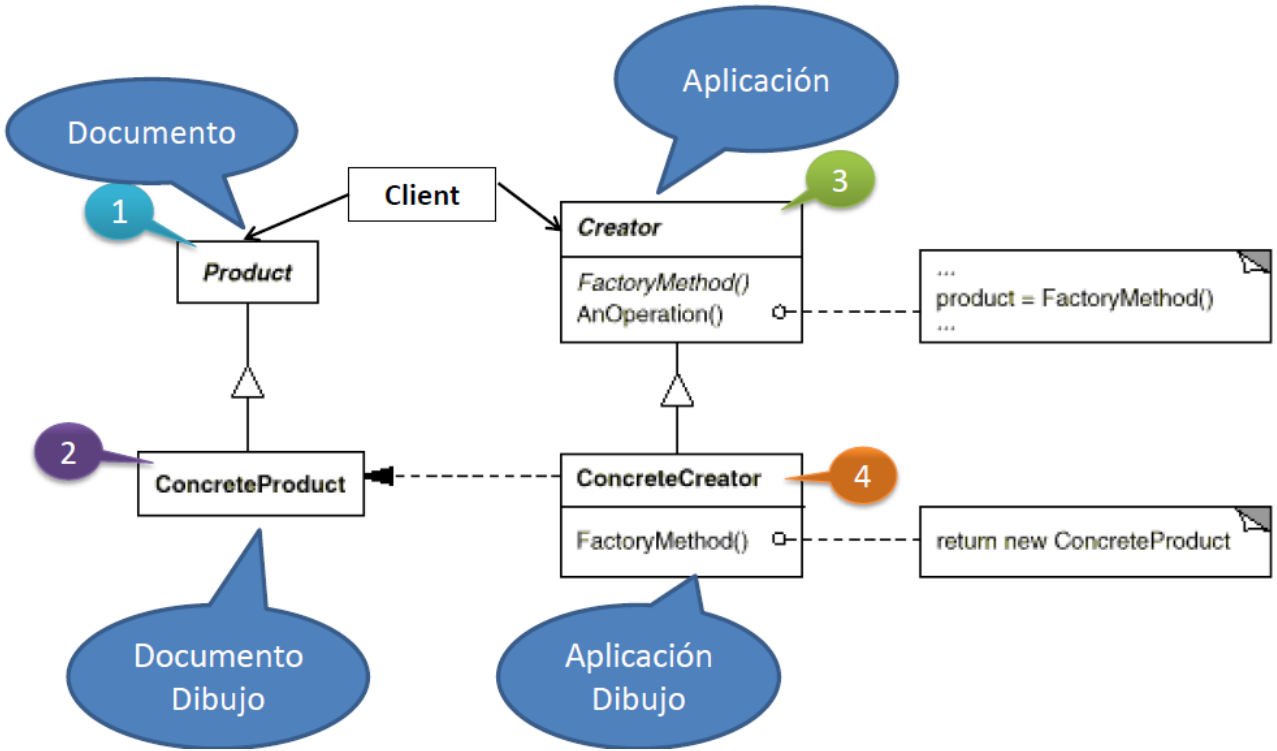
- La clase **Aplicación** no sabe qué tipo de documento específico debe crear, sólo cuándo debe hacerlo.
- Esto implicaría que el framework deberá crear instancias de cada subclase, pero sólo conoce a las clases abstractas (**Aplicación** y **Documento**) y éstas no pueden ser instanciadas.
- Con el patrón Factory Method, las subclases de **Aplicación** redefinirán su operación Abstracta CrearDocumento para devolver la subclase de **Documento** adecuada.
- Con ello, **Aplicación** puede crear instancias de **Documentos** específicos sin conocer sus clases.
- **CrearDocumento** se convertirá en un **FactoryMethod** y fabricará objetos.

Patrón Creacional: Factory Method

- **APLICACIONES**
- Usar cuando:
  - Una clase no puede anticipar la clase (concreta) de objetos que debe crear.
  - Una clase necesita que sus subclases especifiquen los objetos que ésta deba crear.
  - Se cuenta con una familia de clases que comparten atributos y métodos en común pero que lo implementan de manera diferente.
  - Ejemplos de aplicación:
    - Framework de aplicaciones
    - Conexiones a diferentes tipos de bases de datos

Patrón Creacional: Factory Method

- **ESTRUCTURA**



UNIVERSIDAD DE LIMA

Patrón Creacional: Factory Method

- **PARTICIPANTES**
  1. **Product**: Interfaz de los objetos que crea el método de fabricación (**Documento**)
  2. **ConcreteProduct**: Implementa la interfaz Product. Debe existir una clase **ConcreteProduct** por cada tipo de producto que queremos crear (**DocumentoDibujo**)
  3. **Creator (Factory)**: Declara el método de fabricación, el cual devuelve un objeto de tipo **Product**. También puede definir una implementación predeterminada del método de fabricación que devuelva un objeto **ConcreteProduct**. Puede llamar al método de fabricación para crear un objeto **Product**. (**Aplicación**)
  4. **ConcreteCreator** (**Concrete Factory**): Redefine el método de fabricación para crear un objeto **Product**. Debe existir una clase **ConcreteCreator** por cada tipo de producto que queremos crear. (**AplicacionDibujo**)

Patrón Creacional: Factory Method

- **COLABORACIONES**
  - El **Creator** se apoya en sus subclases para definir un método de fabricación de manera que éste devuelva una instancia del **ConcreteProduct** apropiado.

Patrón Creacional: Factory Method

- **EJEMPLO**
  - Tenemos una clase manejadora de conexiones a bases de datos: **DBManager**
  - Tenemos 2 tipos de bases de datos: **MySQL** y **Oracle**.
  - El cliente debería poder crear una conexión a cualquiera de estas bases de datos.
  - Por lo tanto:
    - Creador: DBManager
    - Creador Concreto: CreadorMySQL y Creador ORACLE.
    - Producto: DB
    - Productos Concretos: MYSQL, ORACLE, etc.
    - Cliente

UNIVERSIDAD DE LIMA | 60 años

Patrón Creacional: Abstract Factory

Extensión de Factory Method

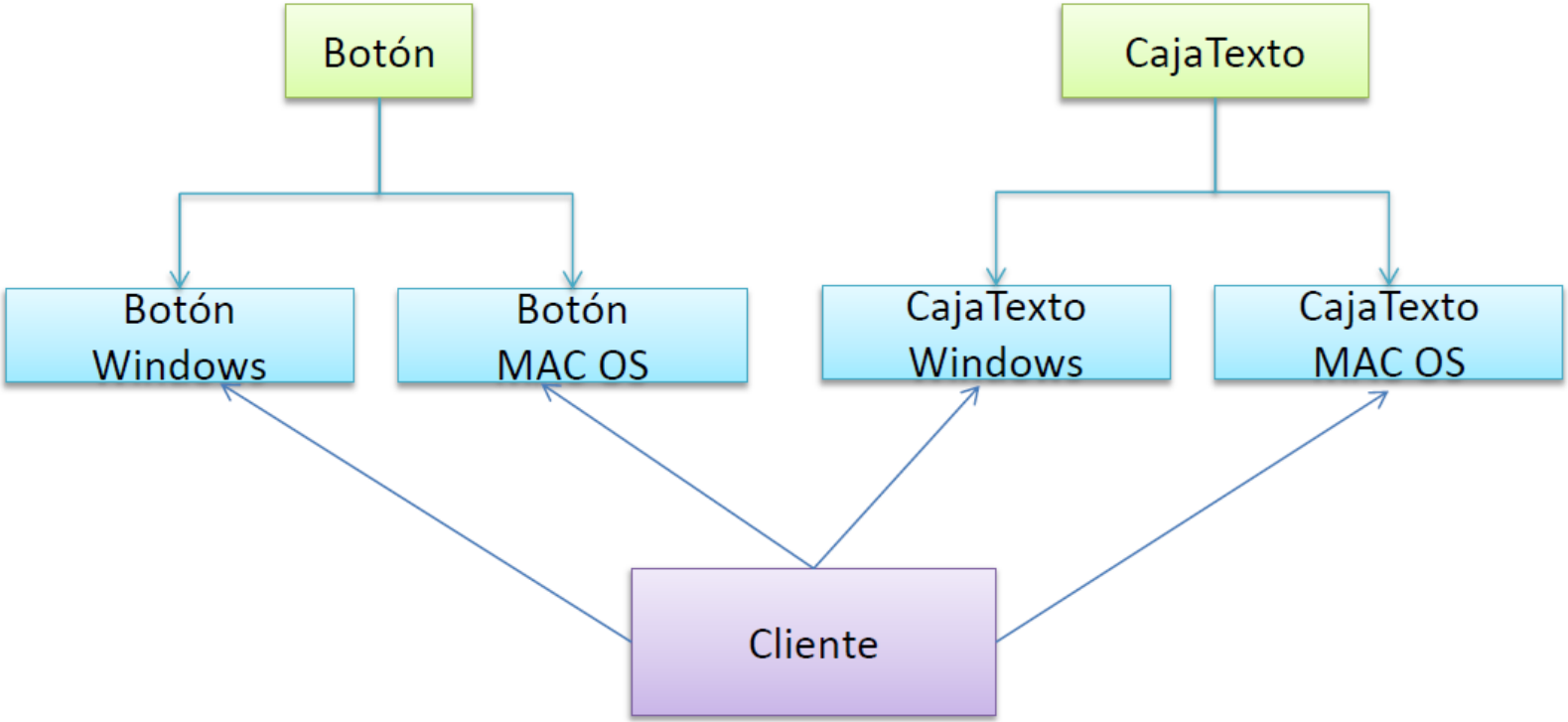
Patrón Creacional: Abstract Factory

- **PROPÓSITO**
  - Define una interfaz para crear familia de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas.

Patrón Creacional: Abstract Factory

- **MOTIVACIÓN**
  - Extensible para futuros estándares
  - Caso de estudio:
    - Su objetivo es soportar múltiples estándares en una aplicación (Windows, Linux, MAC OS) sin necesidad de implementar los componentes de cada estándar en particular. Esto último podría generar la creación de un grupo de clases por cada tipo de estándar existente, generando código complicado de mantener.
    - Tenemos que desarrollar una herramienta multiplataforma para desarrollar interfaces de usuario.
    - Debería permitir crear diversos útiles: botones, cajas de texto, ventanas, etc. Estos útiles tienen diferentes aspectos y formas de comportamiento.
    - Debería ser compatible con diferentes plataformas: Windows, MAC OS.

Patrón Creacional: Abstract Factory



Patrón Creacional: Abstract Factory

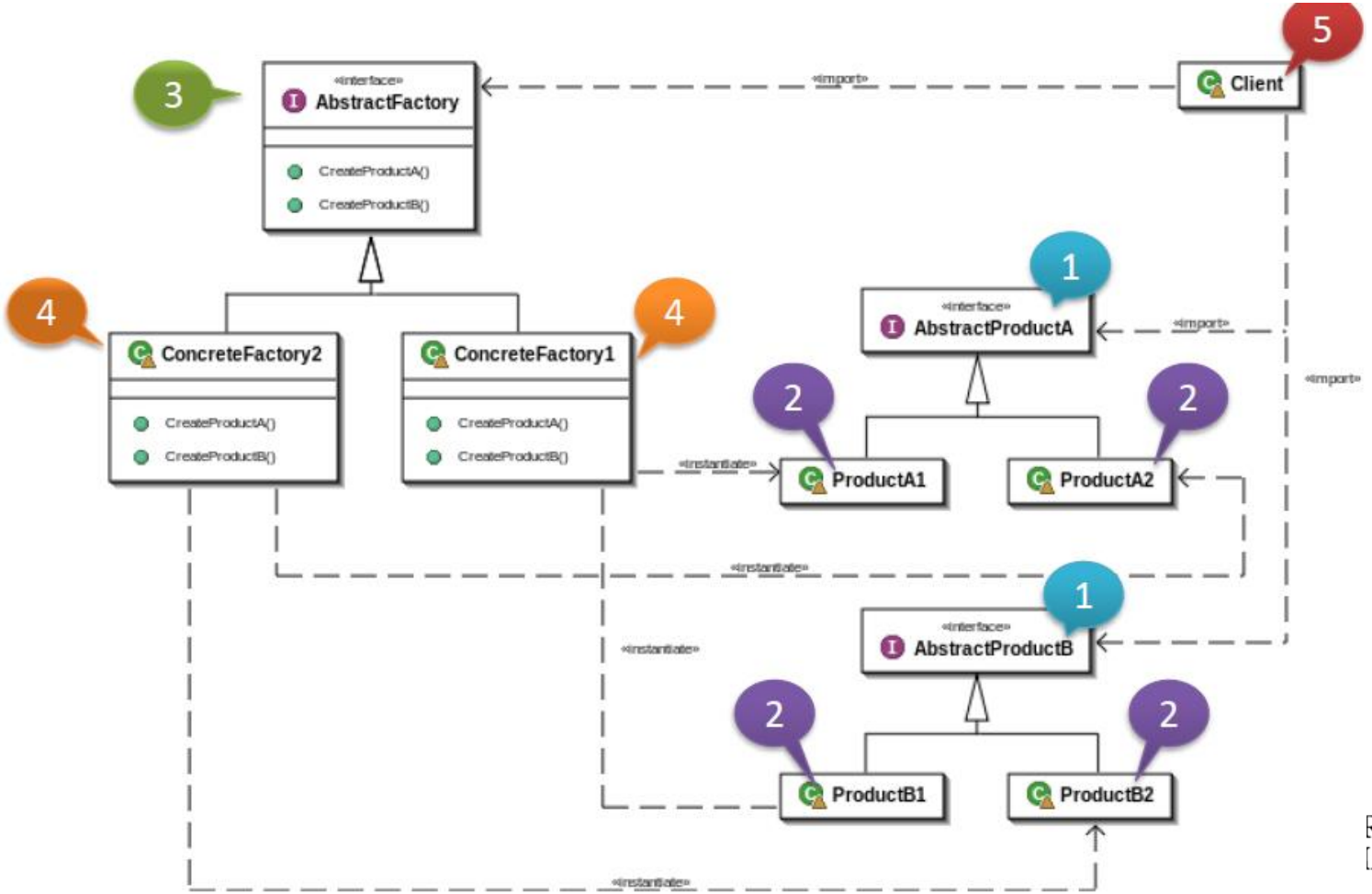
- **APLICACIONES**
  - Un sistema debe ser independiente de los procesos de creación, composición y representación de sus productos.
  - Un sistema debe ser configurado con una familia múltiple de productos.
  - Una familia de productos relacionados se ha diseñado para ser usados conjuntamente (y es necesario reforzar esta restricción).

Patrón Creacional: Abstract Factory

- **PARTICIPANTES**
  1. **AbstractProduct:** Declara una interfaz para un tipo de objeto Product.
  2. **Product:** Define un objeto para que sea creado por la ConcreteFactory correspondiente. Implementa la interfaz AbstractProduct.
  3. **AbstractFactory**: Declara una interfaz para operaciones que crean objetos AbstractProduct.
  4. **ConcreteFactory**: Implementa las operaciones para crear objetos Product.
  5. **Client:** Sólo usa interfaces declaradas por las clases AbstractFactory y AbstractProduct.

Pati

- **E**



RSIDAD
IA

Patrón Creacional: Abstract Factory

- **COLABORACIONES**
  - Normalmente una única instancia de cada **ConcreteFactory** es creada en tiempo de ejecución. Esta **ConcreteFactory** crea productos teniendo una implementación particular. Para crear diferentes productos de objetos, los clientes deben usar diferentes **ConcreteFactory**
  - **AbstractFactory** delega la creación de productos a sus subclases **ConcreteFactory**.

Patrón Creacional: Abstract Factory

- **CONSECUENCIAS (VENTAJAS)**
  - Aísla las clases de implementación: ayuda a controlar los objetos que se creen y encapsula la responsabilidad y el proceso de creación de objetos producto.
  - Hace fácil el intercambio de familias de productos sin mezclarse, permitiendo configurar un sistema con una de entre varias familias de productos:

    cambio de factory -> cambio de familia.
  - Fomenta la consistencia entre productos: Restringe que la aplicación sólo use objetos de una sola familia a la vez.

Patrón Creacional: Abstract Factory

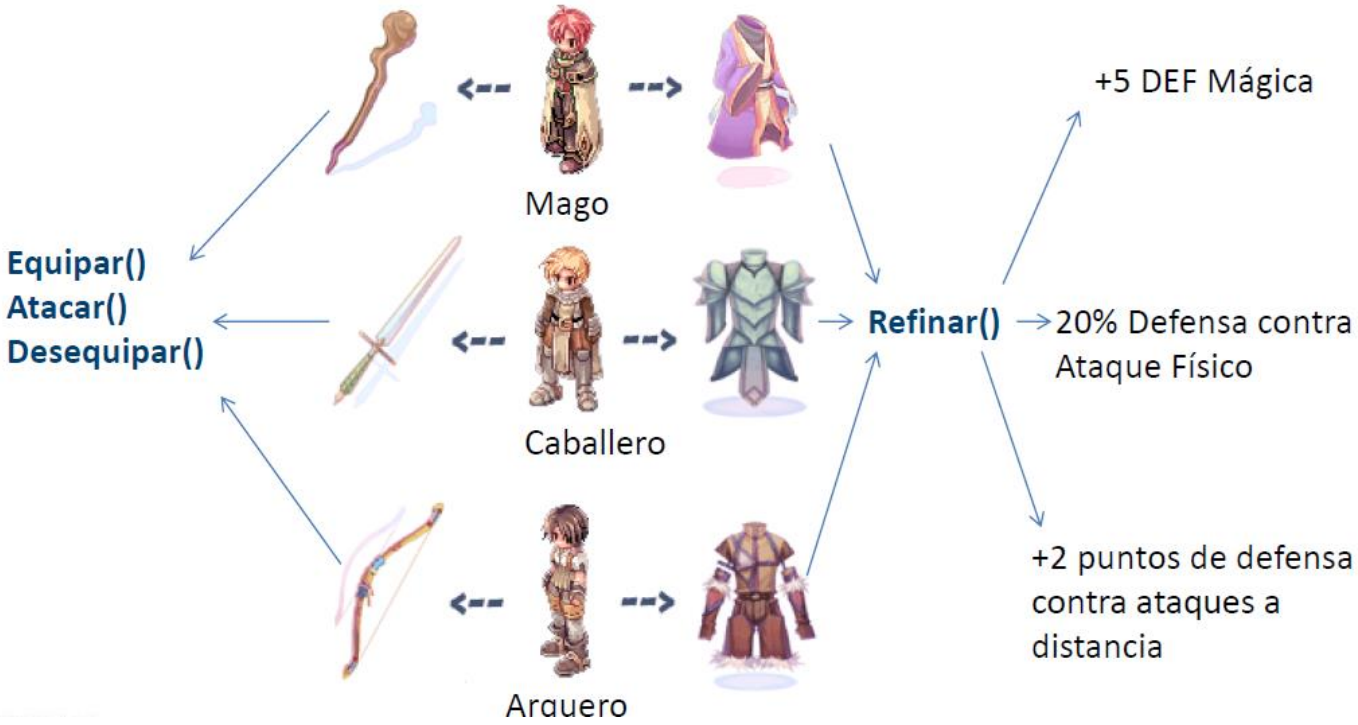
- **CONSECUENCIAS (DESVENTAJAS)**
  - Soportar nuevas clases de productos es difícil. Porque la interfaz del Abstract Factory arregla el bloque de productos que pueden ser creados. Para soportar nuevas clases se requiere extender la interfaz Factory. (cambios en AF clases y subclases)

Patrón Creacional: Abstract Factory

- **EJEMPLO**
  - En los juegos de MMORPG, los jugadores pueden elegir qué tipo de personaje utilizar. Puede ser Mago, Arquero o Caballero.
  - Todos los personajes pueden usar equipos tales como armas y armaduras
  - Las armas tienen las funciones de equiparse, atacar y desequiparse
  - Las armaduras tienen la función de ser refinadas para mejorar su defensa
  - Dependiendo del tipo de personaje, la función del equipo se comporta de manera distinta.

Patrón Creacional: Abstract Factory

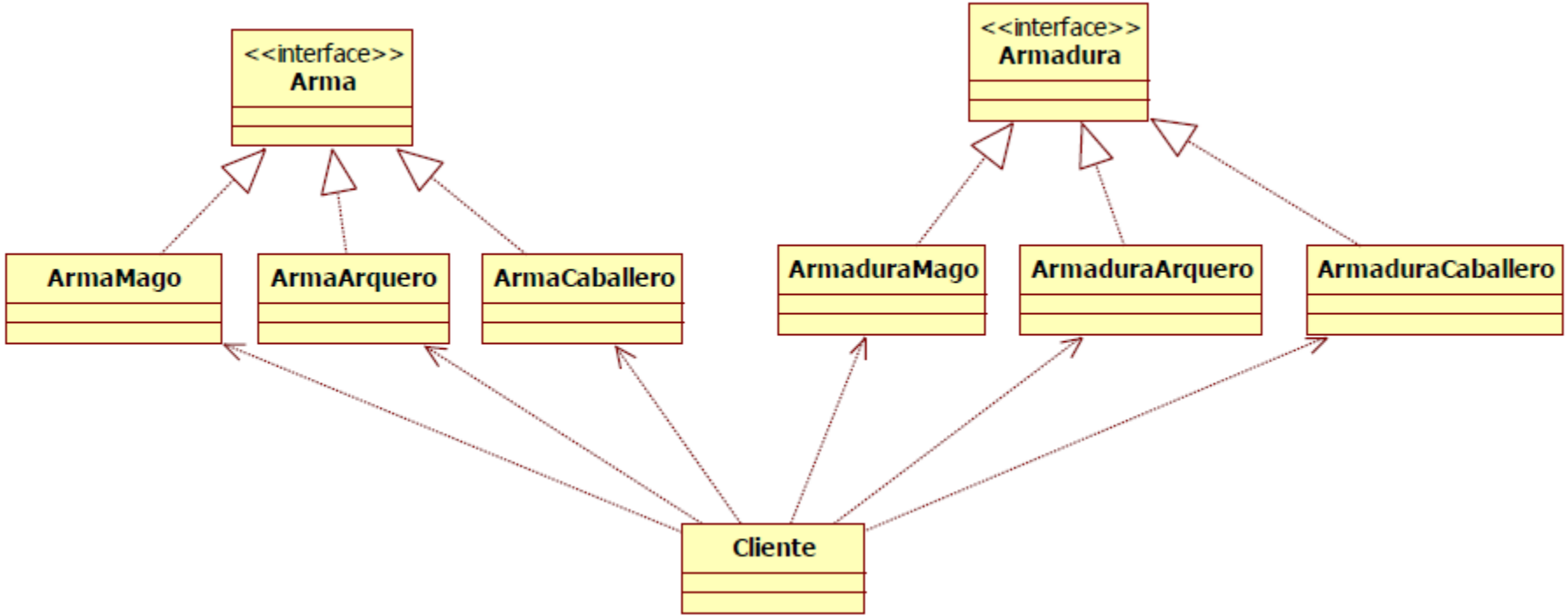
- **EJEMPLO**



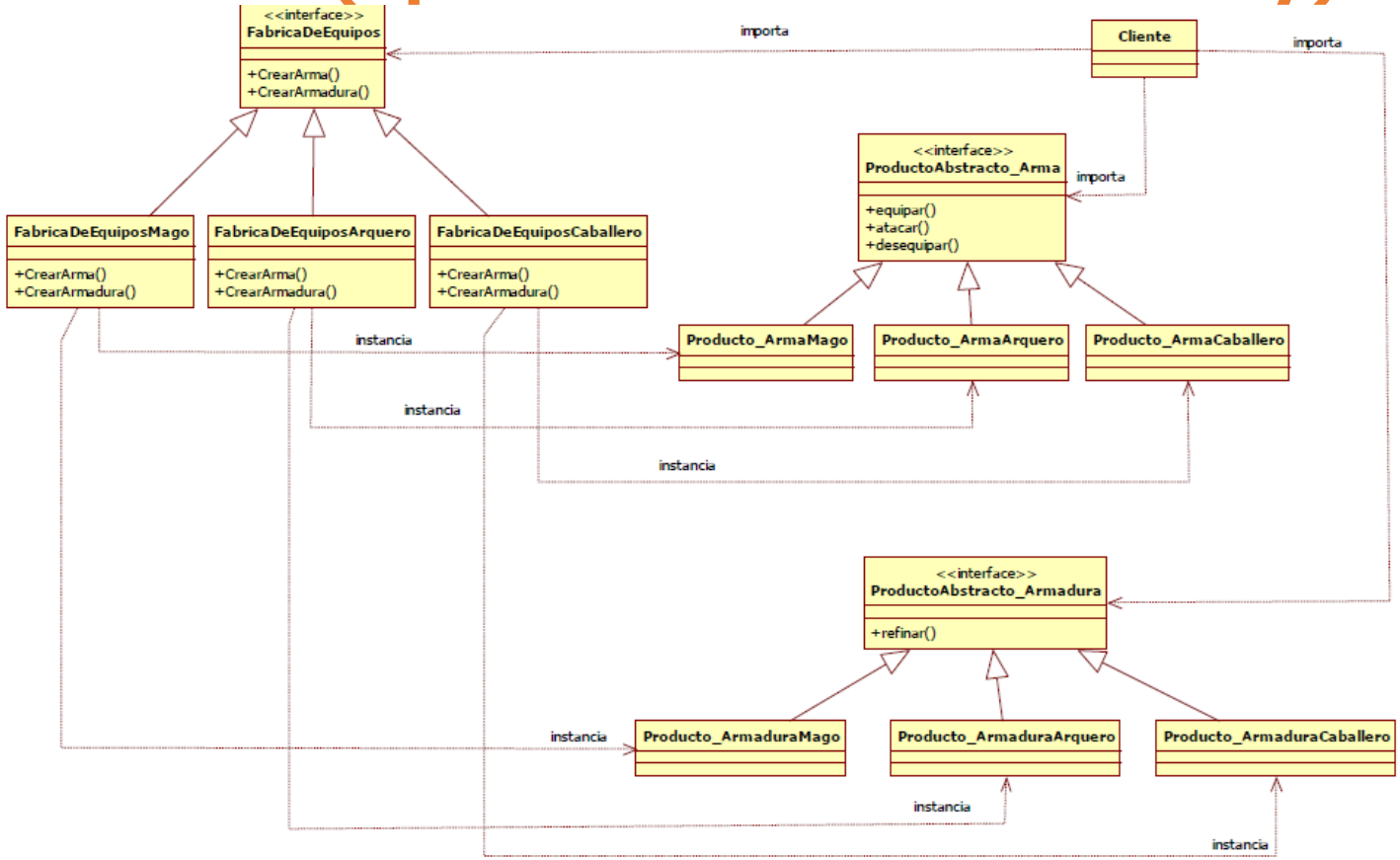
UNIVERSIDAD DE LIMA

Patrón Creacional: Abstract Factory

- **EJEMPLO (actualidad)**



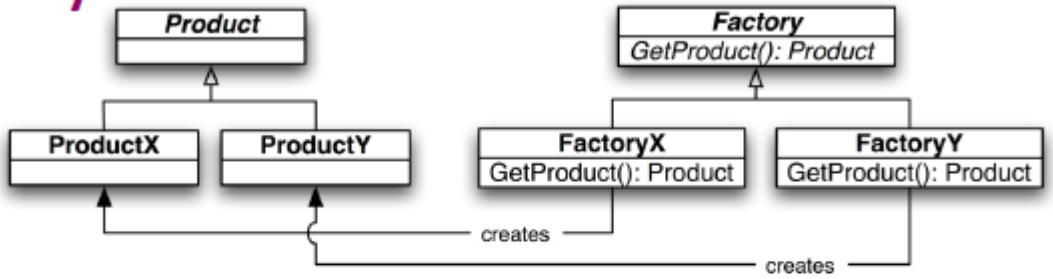
EJEMPLO (aplicando Abstract Factory)



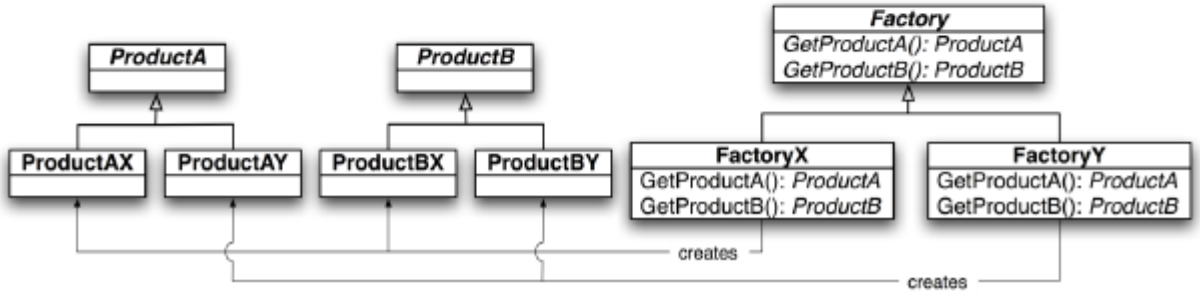
UNIVERSIDAD DE LIMA

Factory Method y Abstract Factory

Factory



Abstract Factory



UNIVERSIDAD DE LIMA

UNIVERSIDAD DE LIMA | 60 años

Patrón Creacional: Singleton

Simple pero bien útil

Patrón Creacional: Singleton

- **PROPÓSITO**
  - Garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella.

Patrón Creacional: Singleton

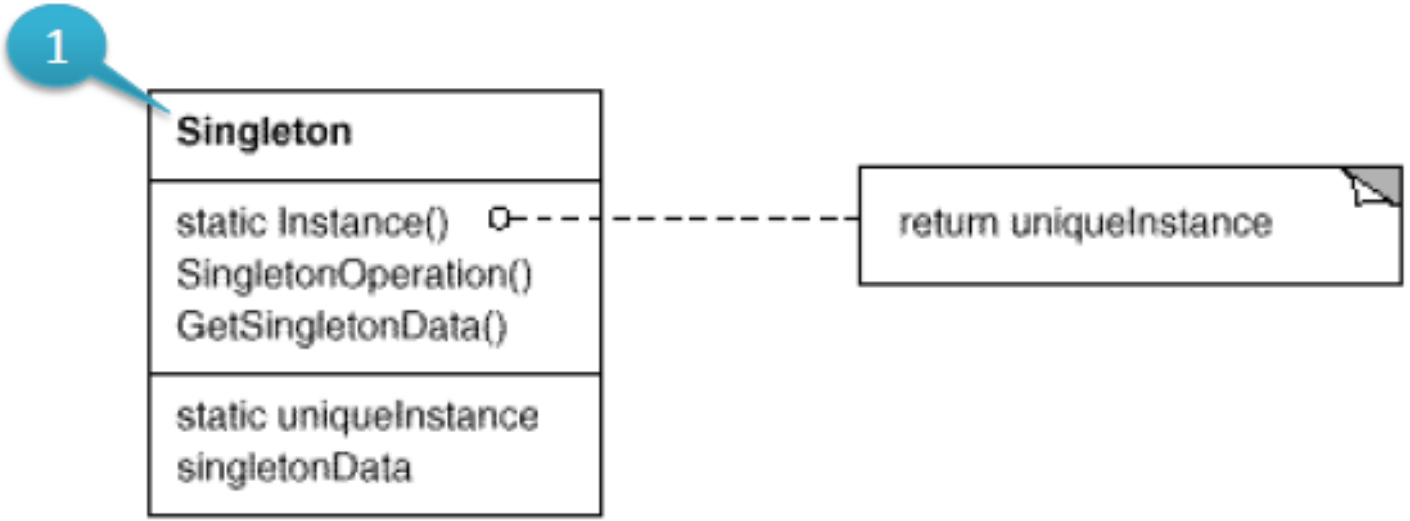
- **MOTIVACIÓN**
  - Dentro de un sistema, existen elementos que deben ser únicos.
  - Por lo general, estas clases (elementos) administran algún recurso de uso común.
  - Por ejemplo: En un reproductor de música, sólo es posible reproducir un audio a la vez. Cuando se desea pasar de un audio a otro, el reproductor debe detener el primero para reproducir el siguiente audio.

Patrón Creacional: Singleton

- **APLICACIONES**
  - Cuando deba existir exactamente una instancia de una clase, y ésta debe ser accesible a los clientes desde un punto de acceso conocido.

Patrón Creacional: Singleton

- **ESTRUCTURA**



Patrón Creacional: Singleton

- **COLABORACIONES**
  1. Singleton:
     - Define una operación INSTANCIA que permite que los clientes accedan a su única instancia.
     - Puede ser responsable de crear esa única instancia.

Patrón Creacional: Singleton

- **IMPLEMENTACIÓN**
  - Cuando se crea un objeto, se invoca al constructor del objeto para que cree una instancia. Por lo general, los constructores son públicos. El Singleton lo que hace es convertir el constructor en privado, de manera que nadie lo pueda instanciar.
  - La instanciación se hace mediante un método público y estático de la clase. En este método se revisa si el objeto ha sido instanciado antes. Si no ha sido instanciado, llama al constructor y guarda el objeto creado en una variable estática del objeto. Si el objeto ya fue instanciado anteriormente, lo que hace este método es devolver la referencia al estado creado anteriormente.

Patrón Creacional: Singleton

- **IMPLEMENTACIÓN**

```
public static Singleton getInstance() {
    if(instance == null) {
      instance = new Singleton();
    }
    return instance;
  }
```

UNIVERSIDAD DE LIMA

Patrón Creacional: Singleton

- **Ejemplo 1: Real–Life Example**
  - Suppose you are a member of a cricket team. And in a tournament your team is going to play against another team. As per the rules of the game, the captain of each side must go for a toss to decide which side will bat (or bowl) first. So, if your team does not have a captain, you need to elect someone as a captain first. And at the same time, your team cannot have more than one captain.
- **Ejemplo 2: Computer World Example**
  - In a software system sometimes we may decide to use only one file system. Usually we may use it for the centralized management of resources.

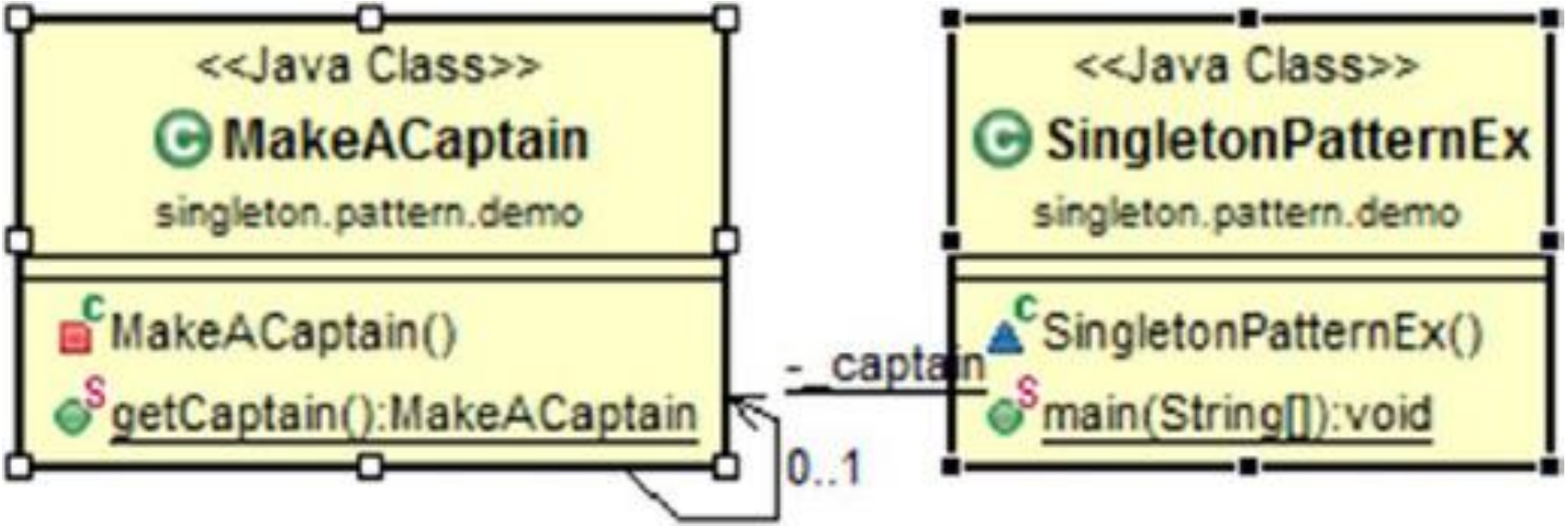
UNIVERSIDAD DE LIMA

Patrón Creacional: Singleton

- **Ejemplo 1**
  - Suppose you are a member of a cricket team. And in a tournament your team is going to play against another team. As per the rules of the game, the captain of each side must go for a toss to decide which side will bat (or bowl) first. So, if your team does not have a captain, you need to elect someone as a captain first. And at the same time, your team cannot have more than one captain.
    - In this example, we have made the constructor private first, so that we cannot instantiate in normal fashion. When we attempt to create an instance of the class, we are checking whether we already have one available copy. If we do not have any such copy, we'll create it; otherwise, we'll simply reuse the existing copy.

Patrón Creacional: Singleton

- **Ejemplo 1**

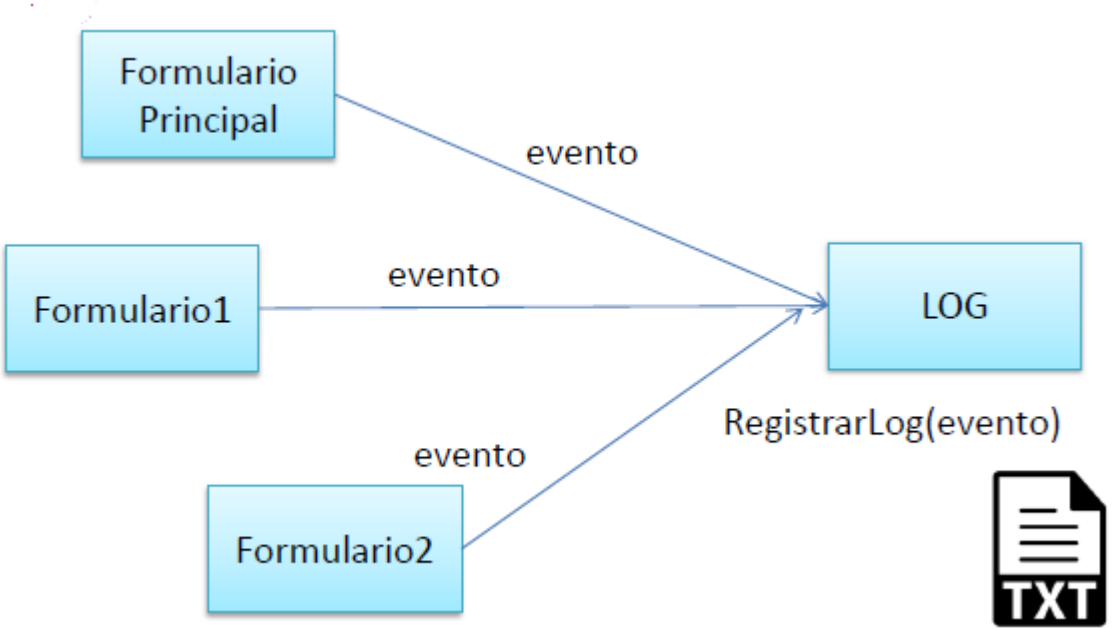


Patrón Creacional: Singleton

- **Ejemplo 3**
  - Tenemos una pequeña aplicación de formularios y botones.
  - Cada acción que se realice sobre las interfaces gráficas (iniciar, abrir ventana, cerrar ventana, clickear botón, etc) genera una línea en un archivo de texto.
  - La clase LOG, posee el método para capturar la hora actual y el evento realizado, para luego almacenarlo en un archivo de texto.
  - Cada formulario captura el evento de cada uno de sus componentes e invoca al método de la clase LOG para escribir en el archivo.

Patrón Creacional: Singleton

- **Ejemplo 3**



Patrón Creacional: Singleton

- **Ejemplo:** CLASE LOG.java (Sin Singleton)

```
public class LogNormal {

    private String mRuta;

    //Método constructor es público
    public LogNormal(){
        mRuta = "log";
    }

    //Metodo utilizado por cada instancia
    public void RegistrarLog(String mensaje){
        String archivo = mRuta + "\\" + Utils.obtenerHoraFechaActual("yyyyMMdd_HH") + ".txt";
        try {
            BufferedWriter out = new BufferedWriter(new FileWriter(archivo, true));
            out.write(Utils.obtenerHoraFechaActual("HH:mm:ss") + ": " + mensaje+ "\r\n");
            out.close();
        } catch (Exception e) {
        }
    }
}
```

Patrón Creacional: Singleton

- **Ejemplo:** Formulario Principal (Sin Singleton)

```
public class FrmPrincipal extends JFrame implements ActionListener {

    //Elementos para el formulario
    private JLabel lblTitulo, lblIndicacion;
    private JButton btnOpcion1;
    private JButton btnOpcion2;

    //Declarar de variable log
    private LogNormal log;   <- Declaro la variable log

    public FrmPrincipal() {
        //Al cerrar la ventana, finalizar el programa
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        //Instanciar la clase LogNormal
        log = new LogNormal(); <- Creo una instancia
        .....
      //Escritura de log
      log.RegistrarLog("El usuario ha iniciado el programa"); <- Registro una línea en el archivo
```

Patrón Creacional: Singleton

- **Ejemplo:** Formulario 1 y 2 (Sin Singleton)

```
public class FrmOpcion2 extends JFrame implements ActionListener {
    //Elementos para el formulario
    private JLabel lblTitulo;
    private JButton boton1,boton2, boton3, boton4;

    //Declarar de variable log
    private LogNormal log;          <- Declaro la variable log

    public FrmOpcion2(){
        //Instanciar la clase LogNormal
        log = new LogNormal();      <- Creo una instancia
   ....
   //Ejecutar acciones de los botones
   public void actionPerformed(ActionEvent e) {
       if (e.getSource() == boton1) {
           //Escritura de log
           log.RegistrarLog("El usuario presiono el Botón 1 de la Ventana 2");
       }if (e.getSource() == boton2) {
           //Escritura de log
           log.RegistrarLog("El usuario presiono el Botón 2 de la Ventana 2");
```

Registro una línea en el archivo por cada Acción que realizo en el Formulario

Patrón Creacional: Singleton

- **Ejemplo:** CLASE LOG.java (Con Singleton)

```
public class LogSingleton {
    private String mRuta;

    //Una sola instancia de la clase Log
    private static LogSingleton mInstanciaLog;

    //Metodo constructor es privado, esto no permite crear una instancia desde otra clase.
    private LogSingleton(){
        mRuta = "log";
    }

    //Metodo estatico para acceder a la instancia única
    public static LogSingleton getInstanciaLog() {
        if(mInstanciaLog==null){
            mInstanciaLog = new LogSingleton();
        }
        return mInstanciaLog;
    }
```

<- Se crea la instancia de la clase Log. Es estática
Porque podrá ser llamada desde cualquier otra clase.

Patrón Creacional: Singleton

- **Ejemplo:** Formularios (Con Singleton)

```
//Escritura de log
LogSingleton.getInstanciaLog().RegistrarLog("El usuario ha iniciado el programa");
```

No debo crear ninguna instancia de Log.
Sólo debo invocar a su método de manera directa en cualquier parte de mi programa

Referencias

1. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software Addison-Wesley.
2. https://github.com/RameshMF/gof-java-design-patterns
3. https://refactoring.guru/design-patterns/abstract-factory/java/example
4. https://refactoring.guru/es/design-patterns/abstract-factory/java/example
5. http://www.cs.sjsu.edu/faculty/pearce/modules/patterns/creational/factory.htm
6. https://www.oscarblancarteblog.com/2014/07/18/patron-de-diseno-factory/