DESARROLLO DE SOFTWARE

Sprint 1

Presentado al tutor(a):

JOHANN LATORRE

Entregado por los estudiantes:

ARLEY GIOVANNI GARCIA NAJAR

Grupo G-29

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA- UNAB

MISIÓN TIC 2022 – CICLO 4

07/11/2021

Un documento en pdf que contenga 2 patrones de diseño explicados con 2 ejemplos de la vida real y con código fuente.

PROTOTYPE

PROPOSITO:

- “Especifica los tipos de objetos a crear usando una instancia prototípica y crea nuevos objetos copiando este prototipo.” Gamma, E., Helm, R., Dr, J. R., & Vlissides, J. (1994).
- “Copia objetos existentes sin que el código dependa de sus clases.” Recfactoring, R. G. (2021).

UTILIDAD

Utilice el patrón Prototype cuando un sistema deba ser independiente de cómo se crean, componen y representan sus productos:

- Y cuando las clases para instanciar se especifican en tiempo de ejecución, por ejemplo, mediante carga dinámica;
- evitar construir una jerarquía de clases de fábricas que sea paralela a la jerarquía de clases de productos;
- cuando las instancias de una clase pueden tener una de las pocas combinaciones diferentes de Estado.

Puede ser más conveniente instalar un número correspondiente de prototipos y clonarlos en lugar de instanciar la clase manualmente, cada vez con el estado apropiado.

PARTICIPANTES

- Prototipo (gráfico): Declara una interfaz para clonarse a sí mismo.
- ConcretePrototype (Staff, WholeNote, HalfNote): Implementa una noperati on para la clonación.
- Cliente (GraphicTool): Crea un nuevo objeto pidiendo a un prototipo que se clone a sí mismo.

PRECAUCIONES EN LA IMPLEMENTACIÓN

Cuando nos disponemos a clonar un objeto es importante tener en cuenta si guarda referencias de otros o no. En este punto hay que distinguir las siguientes formas de replicarlos:

- Copia superficial: el objeto clonado tendrá los mismos valores que el original, guardando también referencias a otros objetos que contenga (por lo que si son modificados desde el objeto original o desde alguno de sus clones el cambio afectará a todos ellos).

- Copia profunda: el objeto clonado tendrá los mismos valores que el original, así como copias de los objetos que contenga el original (por lo que, si son modificados por cualquiera de ellos, el resto no se verán afectados).

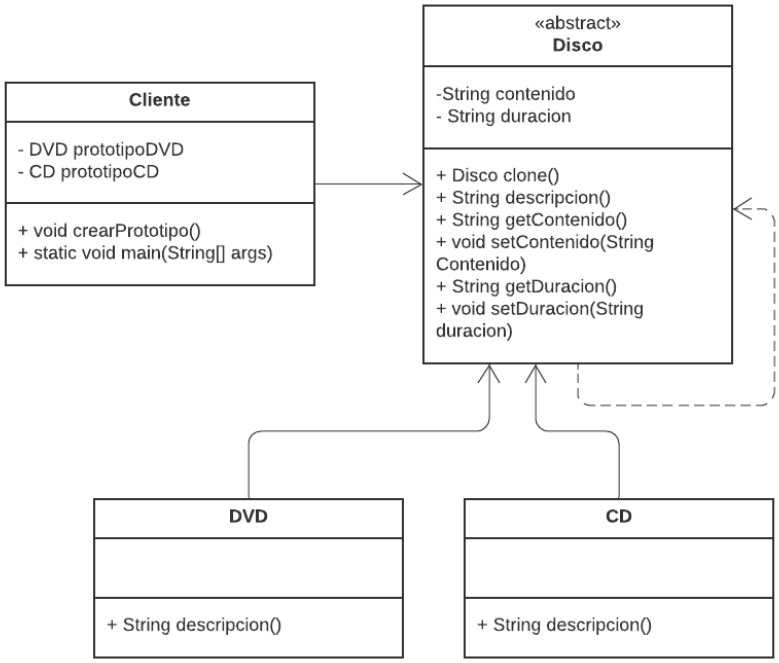
EJEMPLO DEL MUNDO REAL

"Mitosis es el proceso celular por el cual se producen dos núcleos idénticos en preparación para la división celular. En general, la mitosis va seguida inmediatamente del reparto equitativo del núcleo celular, así como del resto del contenido celular en dos células hijas." National Human Genoma Research Institute, N. I. H. (2016).

"Este es un ejemplo de un prototipo que juega un papel activo en copiarse a sí mismo y, por lo tanto, demuestra el patrón de Prototipo. Cuando una célula se divide, resultan dos células de genotipo idéntico. En otras palabras, la célula se clona a sí misma." Franz, B. (2021).

EJEMPLO CON CODIGO

Diseño



Código

```
//Para este ejemplo se utiliza un solo Java Package, de
nombre disco.

package disco;

public abstract class Disco implements Cloneable{

    private String contenido;
    private String duracion;

    public Disco clone() throws CloneNotSupportedException{
        return (Disco) super.clone();
    }
    public abstract String descripcion();

    public String getContenido(){
        return contenido;
    }
    public void setContenido(String contenido){
        this.contenido = contenido;
    }
    public String getDuracion(){
        return duracion;
    }
    public void setDuracion(String duracion){
        this.duracion = duracion;
    }
}

package disco;

public class Cliente{

    private DVD prototipoDVD;
    private CD prototipoCD;

    public void crearProtipo(){
        prototipoDVD = new DVD();
        prototipoDVD.setContenido("Peliculas");
        prototipoDVD.setDuracion("2:30");
        System.out.println("Prototipo:\n " +
prototipoDVD.descripcion() + "\n");
        prototipoCD = new CD();
        prototipoCD.setContenido("Canciones");
```

```
        prototipoCD.setDuracion("1:30");
        System.out.println("Prototipo:\n " +
prototipoCD.descripcion() + "\n");
    }
    public static void main(String[] args) throws
CloneNotSupportedException{
        Cliente fabrica = new Cliente();
        fabrica.crearProtipo();
        Disco cloneCD1 = fabrica.prototipoCD.clone();
        System.out.println("Clone 1: \n" +
cloneCD1.descripcion()+ "\n");

        Disco cloneDVD1 = fabrica.prototipoDVD.clone();
        cloneDVD1.setContenido("Documentales");
        cloneDVD1.setDuracion("1:00");
        System.out.println("Clone 1: \n" +
cloneDVD1.descripcion() + "\n");
    }
}

package disco;

public class DVD extends Disco{

    public String descripcion(){
     return "DVD Contenido: " + this.getContenido() +
"duración: " +  this.getDuracion();
    }
}

package disco;

public class CD extends Disco{

    @Override
    public String descripcion(){
     return "CD Contenido: " + this.getContenido() +
"duración: " +  this.getDuracion();
    }

    public String description() {
        throw new UnsupportedOperationException("Not
supported yet.");
    }
}
```

Resultado en consola:

```
Created dir: C:\Users\Arley\Documents\NetBeansProjects\Disco\build\classes
Created dir: C:\Users\Arley\Documents\NetBeansProjects\Disco\build\empty
Created dir: C:\Users\Arley\Documents\NetBeansProjects\Disco\build\generated-sources\ap-source-output
Compiling 4 source files to C:\Users\Arley\Documents\NetBeansProjects\Disco\build\classes
compile:
run:
Prototipo:
 DVD Contenido: Peliculasduración: 2:30

Prototipo:
 CD Contenido: Cancionesduración: 1:30

Clone 1:
CD Contenido: Cancionesduración: 1:30

Clone 1:
DVD Contenido: Documentalesduración: 1:00

BUILD SUCCESSFUL (total time: 1 second)
```

Ejemplo tomado: “Miguel Ángel (2016)”

El archivo creado en NetBeans para este ejemplo se encuentra cargado en la carpeta que contiene este documento, con el nombre Disco.zip.

PROXY

PROPOSITO:

- “Proporciona un sustituto o marcador de posición para otro objeto para controlar el acceso a él. También conocido como sustituto” Gamma, E., Helm, R., Dr, J. R., & Vlissides, J. (1994).
- “patrón de diseño estructural que te permite proporcionar un sustituto o marcador de posición para otro objeto. Un proxy controla el acceso al objeto original, permitiéndote hacer algo antes o después de que la solicitud llegue al objeto original.” Recfactoring, R. G. (2021).

UTILIDAD

El proxy es aplicable siempre que se necesite una referencia más versátil o sofisticada a un objeto que un simple puntero. A continuación, se muestran varias situaciones comunes en las que se aplica el patrón Proxy:

1. Un proxy remoto proporciona un representante local para un objeto en un espacio de direcciones diferente.
2. Un proxy virtual crea objetos costosos a pedido.
3. Un proxy de protección controla el acceso al objeto original. Los proxies de protección son útiles cuando los objetos deben tener diferentes derechos de acceso.
4. Una referencia inteligente es un reemplazo de un puntero simple que realiza acciones adicionales cuando se accede a un objeto. Los usos típicos incluyen
    - contar el número de referencias al objeto real para que pueda ser
    - se libera automáticamente cuando no hay más referencias (también llamados punteros inteligentes).
    - cargar un objeto persistente en la memoria cuando se hace referencia a él por primera vez.
    - comprobar que el objeto real está bloqueado antes de acceder a él para asegurarse de que ningún otro objeto pueda cambiarlo.

PARTICIPANTES

- Proxy (ImageProxy): - mantiene una referencia que deja al proxy acceder al sujeto real. Proxy puede referirse a un sujeto si las interfaces RealSubject y Subject son las mismas.
    - proporciona una interfaz idéntica a la del sujeto para que un proxy pueda sustituir al sujeto real.
    - controla el acceso al tema real y puede ser responsable de crearlo y eliminarlo.
    - otras responsabilidades dependen del tipo de apoderado:
        - proxy remoto: responsable de codificar una petición y sus argumentos, y de enviarla al objeto remoto.

  - proxy virtual: puede hacer caché de información del objeto real para diferir en lo posible el acceso a este.
  - proxy de protección: comprueba que el cliente tiene los permisos necesarios para realizar la petición.
- Asunto (gráfico): define la interfaz común para RealSubject y Proxy para que un Proxy se puede usar en cualquier lugar donde se espere un RealSubject.
- RealSubject (imagen): define el objeto real que representa el proxy.

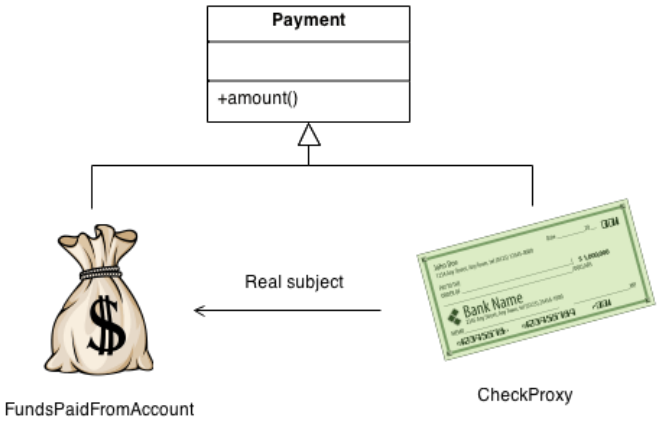
PRECAUCIONES EN LA IMPLEMENTACIÓN

El patrón Proxy introduce un nivel de indirección al acceder a un objeto. La indirección adicional tiene muchos usos, según el tipo de proxy:

1. Un proxy remoto puede ocultar el hecho de que un objeto reside en un espacio de direcciones diferente.
2. Un proxy virtual puede realizar optimizaciones como la creación de un objeto bajo demanda.
3. Tanto los proxies de protección como las referencias inteligentes permiten tareas de limpieza adicionales cuando se accede a un objeto.

Hay otra optimización que el patrón Proxy puede ocultar al cliente. Se llama copia en escritura y está relacionado con la creación bajo demanda.

EJEMPLO DEL MUNDO REAL

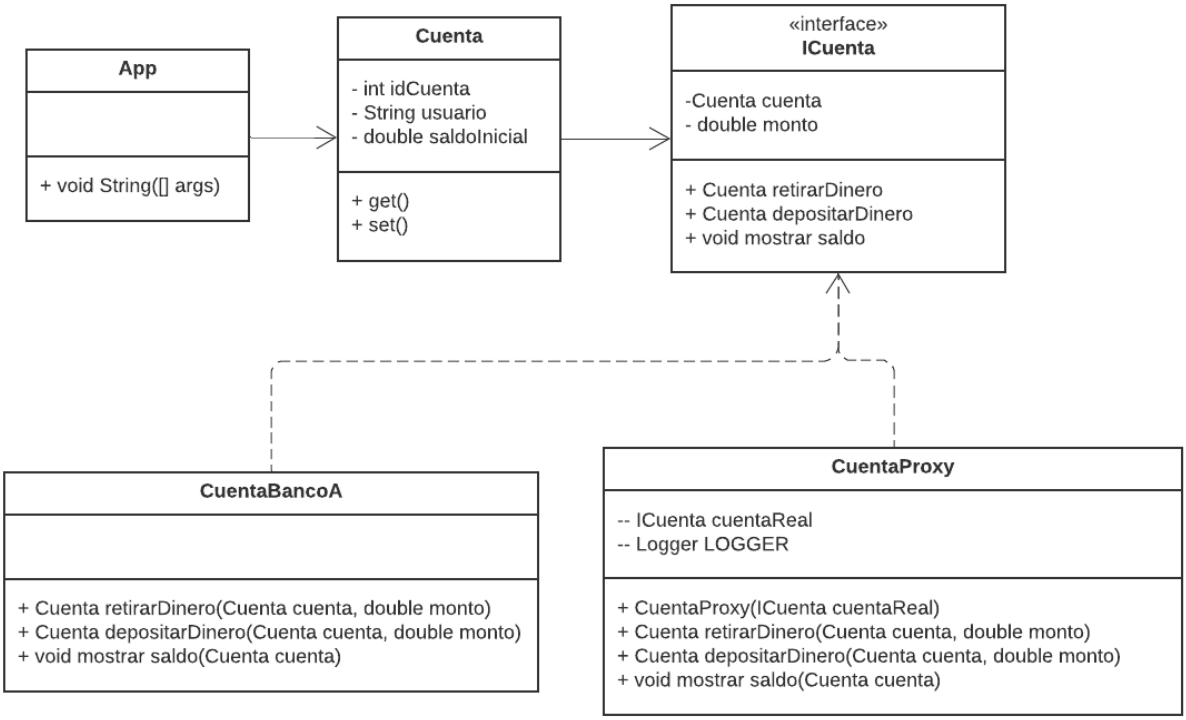


"En este caso "El proxy proporciona un sustituto o marcador de posición para proporcionar acceso a un objeto. Un cheque o giro bancario es un sustituto de los fondos en una cuenta. Se puede utilizar un cheque en lugar de efectivo para realizar compras y, en última instancia, controla el acceso al efectivo en la cuenta del emisor." Source Making, S. M. (2021).

"Mediante la implementación del patrón de diseño Proxy se crea un mecanismo de seguridad, el cual intercepta las ejecuciones de procesos para validar si el usuario que intenta ejecutar cuenta con los privilegios necesarios, evitando que usuarios no autorizados los ejecuten, además, una vez que el proceso es ejecutado, se auditara la ejecución y quedará un registro de la ejecución. Todo esto se realizará sin que el usuario se dé cuenta, pues el proxy envolverá la lógica de seguridad." Blancarte, O. (2020).

EJEMPLO CON CODIGO

Diseño



Código

```
//Para este ejemplo se utiliza un solo Java Package, de
nombre Banco_proxy.

package disco;

package banco_proxy;

public class App {

    public static void main(String[] args) {
        Cuenta c = new Cuenta(1, "misionTic", 100);
        ICuenta cuentaProxy = new CuentaProxy(new
CuentaBancoA());
        cuentaProxy.mostrarSaldo(c);
        c = cuentaProxy.depositarDinero(c, 50);
        c = cuentaProxy.retirarDinero(c, 20);
        cuentaProxy.mostrarSaldo(c);
      }
}
```

```
package banco_proxy;

public class Cuenta {
    private int idCuenta;
    private String usuario;
    private double saldoInicial;

    public Cuenta(int idCuenta, String usuario, double
saldoInicial) {
        this.idCuenta = idCuenta;
        this.usuario = usuario;
        this.saldoInicial = saldoInicial;
    }

    public int getIdCuenta() {
        return idCuenta;
    }

    public void setIdCuenta(int idCuenta) {
        this.idCuenta = idCuenta;
    }

    public String getUsuario() {
        return usuario;
    }

    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }

    public double getSaldoInicial() {
        return saldoInicial;
    }

    public void setSaldoInicial(double saldoInicial) {
        this.saldoInicial = saldoInicial;
    }
}


package banco_proxy;

public class CuentaBancoA implements ICuenta{

    public Cuenta retirarDinero(Cuenta cuenta, double monto)
{
```

```
        double saldoActual = cuenta.getSaldoInicial() -
monto;
        cuenta.setSaldoInicial(saldoActual);
        System.out.println("Saldo actual: " +
cuenta.getSaldoInicial());
        return cuenta;
    }

    public Cuenta depositarDinero(Cuenta cuenta, double
monto) {
        double saldoActual = cuenta.getSaldoInicial() +
monto;
        cuenta.setSaldoInicial(saldoActual);
        System.out.println("Saldo actual: " +
cuenta.getSaldoInicial());
        return cuenta;
    }

    public void mostrarSaldo(Cuenta cuenta) {
        System.out.println("Saldo actual: " +
cuenta.getSaldoInicial());
    }

}


package banco_proxy;

import java.util.logging.Logger;

public class CuentaProxy implements ICuenta{

    private ICuenta cuentaReal;
    private final static Logger LOGGER =
Logger.getLogger(CuentaProxy.class.getName());

    public CuentaProxy(ICuenta cuentaReal) {
            this.cuentaReal = cuentaReal;
    }

    @Override
    public Cuenta retirarDinero(Cuenta cuenta, double monto)
{
        LOGGER.info("----Cuenta Proxy - Retirar Dinero----");
        if (cuentaReal == null) {
                cuentaReal = new CuentaBancoA();
```

```
                return cuentaReal.retirarDinero(cuenta,
monto);
        } else {
                return cuentaReal.retirarDinero(cuenta,
monto);
        }
    }

    @Override
    public Cuenta depositarDinero(Cuenta cuenta, double
monto) {
        LOGGER.info("----Cuenta Proxy - Depositar Dinero----
");
        if (cuentaReal == null) {
                cuentaReal = new CuentaBancoA();
                return cuentaReal.depositarDinero(cuenta,
monto);
        } else {
                return cuentaReal.depositarDinero(cuenta,
monto);
        }
    }

    @Override
    public void mostrarSaldo(Cuenta cuenta) {
        LOGGER.info("----Cuenta Proxy - Mostrar Dinero----");
        if (cuentaReal == null) {
                cuentaReal = new CuentaBancoA();
                cuentaReal.mostrarSaldo(cuenta);
        } else {
                cuentaReal.mostrarSaldo(cuenta);
        }
    }
}

package banco_proxy;

public interface ICuenta {
      Cuenta retirarDinero(Cuenta cuenta, double monto);
      Cuenta depositarDinero(Cuenta cuenta, double monto);
      void mostrarSaldo(Cuenta cuenta);
}
```

Resultado en consola:

```
Deleting: C:\Users\Arley\Documents\NetBeansProjects\Banco_Proxy\build\built-jar.properties
deps-jar:
Updating property file: C:\Users\Arley\Documents\NetBeansProjects\Banco_Proxy\build\built-jar.properties
Compiling 1 source file to C:\Users\Arley\Documents\NetBeansProjects\Banco_Proxy\build\classes
compile-single:
run-single:
nov. 12, 2021 2:51:59 P. M. banco_proxy.CuentaProxy mostrarSaldo
INFO: ----Cuenta Proxy - Mostrar Dinero----
Saldo actual: 100.0
Saldo actual: 150.0
Saldo actual: 130.0
nov. 12, 2021 2:51:59 P. M. banco_proxy.CuentaProxy depositarDinero
Saldo actual: 130.0
INFO: ----Cuenta Proxy - Depositar Dinero----
nov. 12, 2021 2:51:59 P. M. banco_proxy.CuentaProxy retirarDinero
INFO: ----Cuenta Proxy - Retirar Dinero----
nov. 12, 2021 2:51:59 P. M. banco_proxy.CuentaProxy mostrarSaldo
INFO: ----Cuenta Proxy - Mostrar Dinero----
BUILD SUCCESSFUL (total time: 1 second)
```

Ejemplo tomado: “MitoCode. (2018)”

El archivo creado en NetBeans para este ejemplo se encuentra cargado en la carpeta que contiene este documento, con el nombre Banco_Proxy.zip.

REFERENCIAS

National Human Genoma Research Institute, N. I. H. (2016, 19 octubre). *Mitosis | NHGRI*. Genome.gov. Recuperado 8 de noviembre de 2021, de https://www.genome.gov/es/genetics-glossary/Mitosis

Franz, B. (2021). *1: Primer patrÃ³n de diseÃ±o - FranzBurneo/Patrones-de-dise-o Wiki*. git-hub. Recuperado 8 de noviembre de 2021, de https://github-wiki-see.page/m/FranzBurneo/Patrones-de-dise-o/wiki/1:-Primer-patr%C3%B3n-de-dise%C3%B1o

Prototipo (patrón de diseño). (2021, 24 julio). En *Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/wiki/Prototipo_(patr%C3%B3n_de_dise%C3%B1o)

Miguel Angel. (2016, 2 diciembre). *Patrón de Diseño Prototype* [Vídeo]. YouTube. https://www.youtube.com/watch?v=u52Xaa2HAAY&t=6s

Source Making, S. M. (2021, 12 noviembre). *Design Patterns and Refactoring*. Sourcemaking. Recuperado 12 de noviembre de 2021, de https://sourcemaking.com/design_patterns/proxy

Blancarte, O. (2020, 26 septiembre). *Proxy*. reactiveprogramming.io. Recuperado 11 de noviembre de 2021, de https://reactiveprogramming.io/blog/es/patrones-de-diseno/proxy

Proxy (patrón de diseño). (2020, 10 noviembre). En *Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/wiki/Proxy_(patr%C3%B3n_de_dise%C3%B1o)

Recfactoring, R. G. (2021, 12 noviembre). *Patrones de diseño / Design patterns*. refactoring.guru. Recuperado 12 de noviembre de 2021, de https://refactoring.guru/es/design-patterns

Gamma, E., Helm, R., Dr, J. R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software* (1.ª ed.). Addison-Wesley Professional.

MitoCode. (2018, 29 julio). *Curso de Patrones de diseño - 8 Proxy* [Vídeo]. YouTube. https://www.youtube.com/watch?v=LUJbqdthTzA