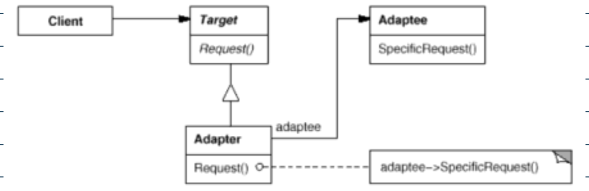
Patrones de diseño- RESUMEN

martes, 5 de marzo de 2024 13:40

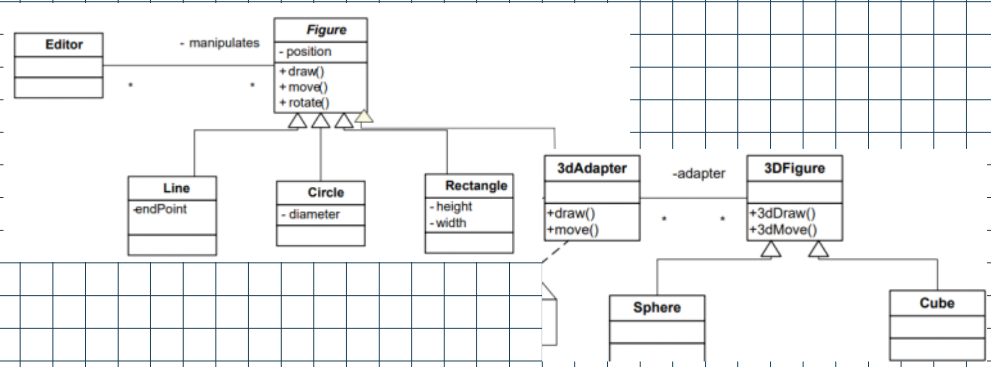
Adapter

USO: convertir la interfaz de una clase en otra que el cliente espere. Permite que varias clases trabajen juntas cuando tiene n métodos con distinto nombre. Se usa cuando tenemos una interfaz y una clase con nombres de métodos distintos y necesitamos hacer que anden juntos.



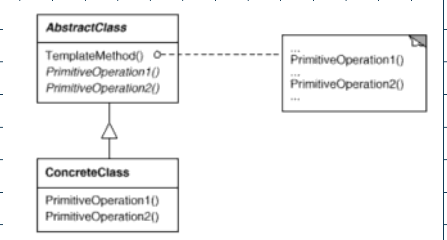
- Target: Define la interfaz especifica que usa el cliente
- Client: Colabora con objetos que satisfacen la interfaz de Target
- Adaptee: Define una interfaz que precisa ser adaptada
- Adapter: Adapta la interfaz del Adaptee a la interfaz de Target

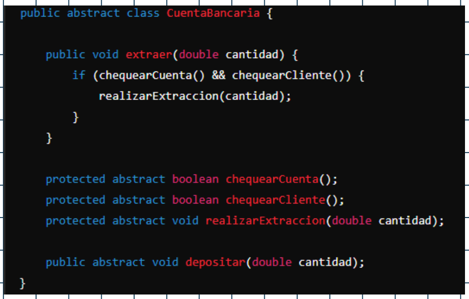
EJEMPLO:



Template method

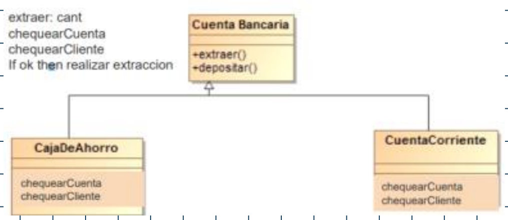
USO: Se usa cuando tenemos una clase padre a la que se le puede dar parte de la estructura de un método pero cuyos hijos tien en algunos pasos diferentes entre sí. La idea es que las subclases redefinan algunas cosas sin cambiar la estructura.
Es como que se sacan las cosas especificas a cada objeto y se deja el comportamiento general en el padre. Los métodos resulta ntes de los hijos van a ser más primitivos.





```
public abstract class CuentaBancaria {

    public void extraer(double cantidad) {
        if (chequearCuenta() && chequearCliente()) {
            realizarExtraccion(cantidad);
        }
    }

    protected abstract boolean chequearCuenta();
    protected abstract boolean chequearCliente();
    protected abstract void realizarExtraccion(double cantidad);

    public abstract void depositar(double cantidad);
}
```

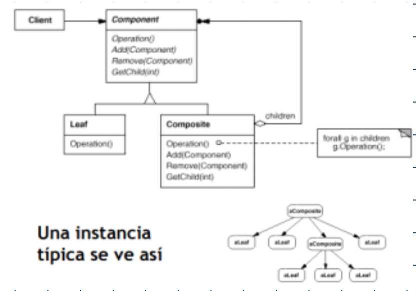
EJEMPLO:



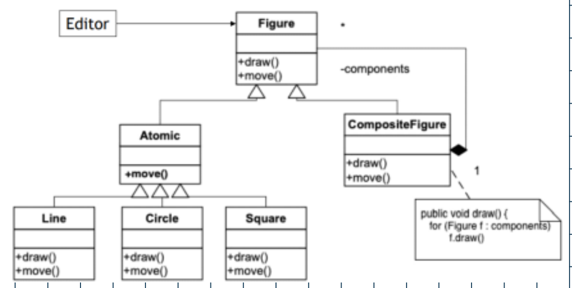
Pattern Composite

USO: Manejar objetos compuestos y atómicos uniformemente.
Su estructura es la de un árbol.
Está compuesto por:
    Component: El cual declara la interfaz de la composición y como definir y acceder a los "hijos".
    Leaf(hoja): Representan las hojas del árbol, las cuales no tienen hijos.
    Composite: Define el comportamiento de los "hijos", e implementa las operaciones para manejar los mismos.

Es para tener un grupo de objetos y que todos respondan al mismo llamado. Pero no solo sirven para un grupo de estos, sino qu e también se puede utilizar para uno solo, lo que serían las hojas.
Ejemplo: si Tenes un cuadrado y un triángulo, ambos pueden reaccionar a pintar() y dibujar(), ahora si una casa está compuest a por un triángulo y un cuadrado, esta casa sería un Composite, el cual gracias al Component, podrá entender y saber cómo llamar a los métodos, logrando dibujar el cuadrado y el triángulo, y hasta poder pintarlos.

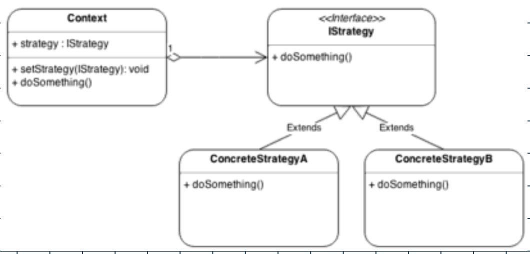


EJEMPLO:

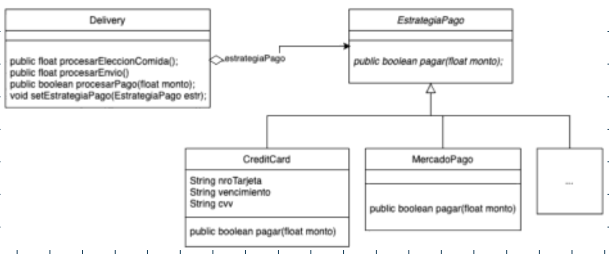


Strategy

USO: Se usa cuando tengo que aplicar algoritmos distintos para hacer una misma tarea. Lo que se hace es meter cada algoritmo en un objeto. Se define una familia de algoritmos, encapsular cada uno y hacerlos intercambiables. El Strategy permite que el algoritmo varíe independientemente de los clientes que lo usan.

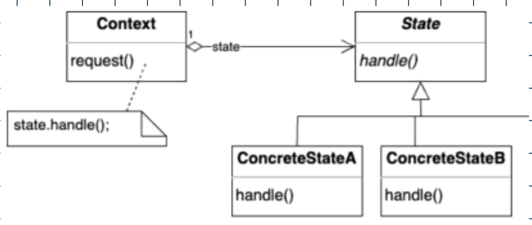


EJEMPLO:

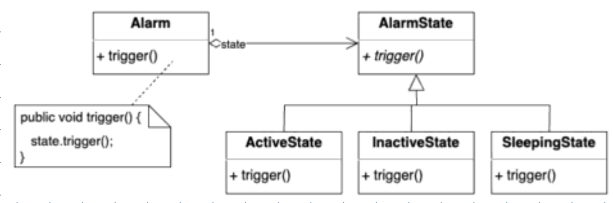

Delivery
public float procesarEleccionComida();
public float procesarEnvio()
public boolean procesarPago(float monto);
void setEstrategiaPago(EstrategiaPago estr);
estrategiaPago
EstrategiaPago
public boolean pagar(float monto);
CreditCard
String nroTarjeta
String vencimiento
String cvv
public boolean pagar(float monto)
MercadoPago
public boolean pagar(float monto)
...

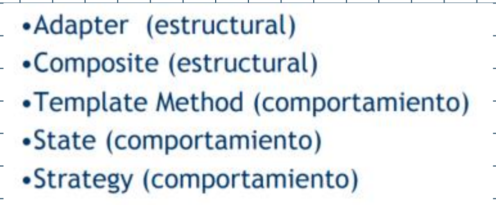
STATE

USO: Parecido al de arriba pero lo voy a usar cuando el comportamiento de un objeto depende del estado en el que está. Se tie nen que como comunicar entre los estados.

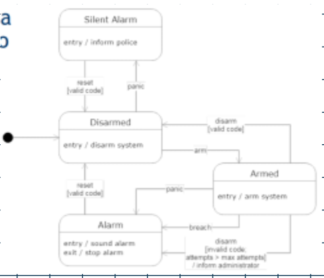


EJEMPLO:



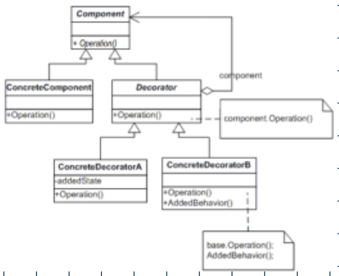

- Adapter (estructural)
- Composite (estructural)
- Template Method (comportamiento)
- State (comportamiento)
- Strategy (comportamiento)

a



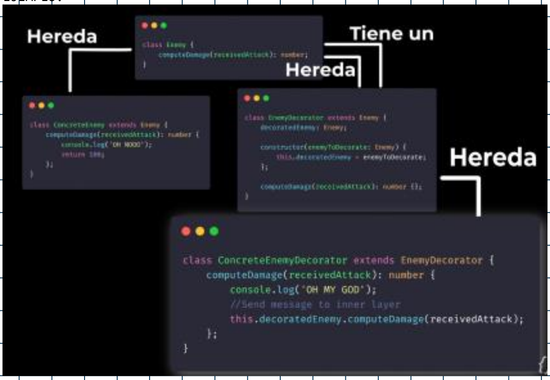
Decorator

USO: agregar y sacar funciones dinámicamente sin complicarse la vida y tener jerarquías enormes.



Un decorator siempre es abstract.
Un component normalmente es una interface// puede llegar a ser un abstract.

EJEMPLO:

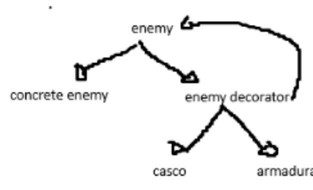


```
class Enemy {
    computeDamage(receivedAttack): number;
}
```

```
class ConcreteEnemy extends Enemy {
    computeDamage(receivedAttack): number {
        console.log('OH NOOO');
        return 100;
    };
}
```

```
class EnemyDecorator extends Enemy {
    decoratedEnemy: Enemy;

    constructor(enemyToDecorate: Enemy) {
        this.decoratedEnemy = enemyToDecorate;
    };

    computeDamage(receivedAttack): number {};
}
```

```
class ConcreteEnemyDecorator extends EnemyDecorator {
    computeDamage(receivedAttack): number {
        console.log('OH MY GOD');
        //Send message to inner layer
        this.decoratedEnemy.computeDamage(receivedAttack);
    };
}
```



Acá es como que se van acumulando los tipos de armadura que tiene un enemigo: si tieme casco recibe la mitad, si tiene armadu ra recibe lo que recibió -1.

OJO, cambia como vas armando el objeto

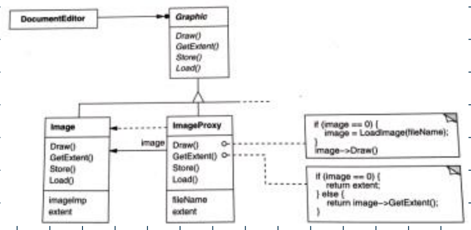
```
let enemy = new DifficultEnemy();
let enemyWithHelmet = new HelmetDecorator(enemy);
let enemyWithHelmetAndArmour = new ArmourDecorator(enemyWithHelmet);

let computedDamaged = enemyWithHelmetAndArmour.takeDamage();
```

Acá primero el base devuelve el daño, después va a aplicar el daño de helmet y de ahí se llama al de armor.

Proxy

USO: es un intermediario. El proxy tiene una instancia del objeto referenciado(el objeto que está siendo "proxiado"), saben c ontestar los mismos métodos, pero el proxy puede implementar lógica adicional para hacer lo mismo (un método de diferente manera si se necesita).

EJEMPLO:

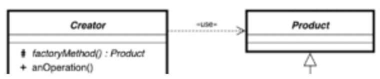


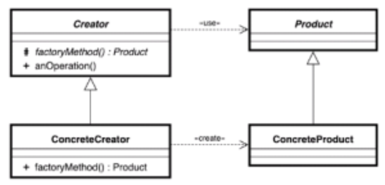
Cargar las imágenes bajo demanda, utilizando un objeto proxy. El proxy se comporta como una imagen normal y es el responsable de cargar la imagen bajo demanda
Con eso si la imagen todavía no está se carga temporalmente un placeholder digamos, como para indicarle al user que ahí va a haber una imagen.

Factory Method

USO: es un strategy pero creacional. La idea es poder hacer un objeto constructor que se ocupe de implementar la lógica de cr ear los objetos por separado. Esto nos permite cambiar cómo los creamos de una manera rápida

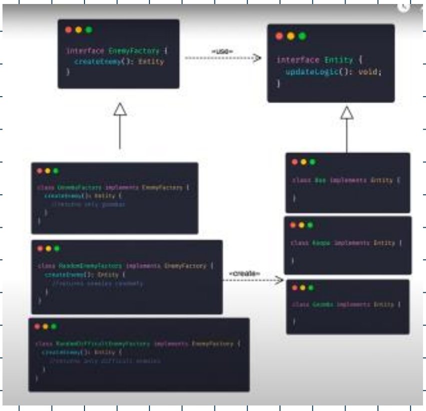
- Cuando la creación del objeto implica una lógica compleja y no solo una simple instanciación.
- Cuando la clase no puede anticipar qué clase de objetos debe crear.
- Cuando se desea delegar la creación de objetos a subclases específicas.





- Cuando la clase no puede anticipar qué clase de objetos debe crear.
- Cuando se desea delegar la creación de objetos a subclases específicas.

EJEMPLO: Enemy factory es interfaz de las distintas formas de crear enemigos.
Vos haces un:
    EnemyFactory enemFactory= new randomEnemyFactory() y después enemFactory.createEnemy()
Así, si quiero cambiar la estrategia de creación cambio el creador y punto.



```
interface EnemyFactory {
  createEnemy(): Entity
}
```

```
interface Entity {
  updateLogic(): void;
}
```

```
class GoombaFactory implements EnemyFactory {
  createEnemy(): Entity {
    //returns only goombas
  }
}
```

```
class RandomEnemyFactory implements EnemyFactory {
  createEnemy(): Entity {
    //returns enemies randomly
  }
}
```

```
class RandomDifficultEnemyFactory implements EnemyFactory {
  createEnemy(): Entity {
    //returns only difficult enemies
  }
}
```

```
class Bee implements Entity {
}
```

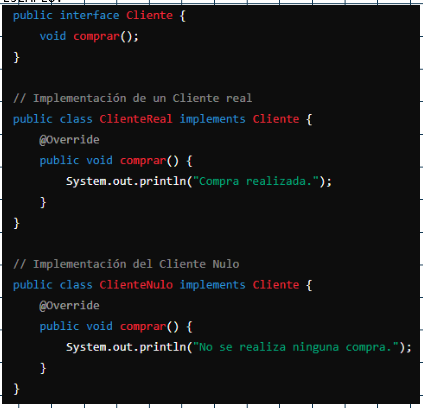
```
class Koopa implements Entity {
}
```

```
class Goomba implements Entity {
}
```

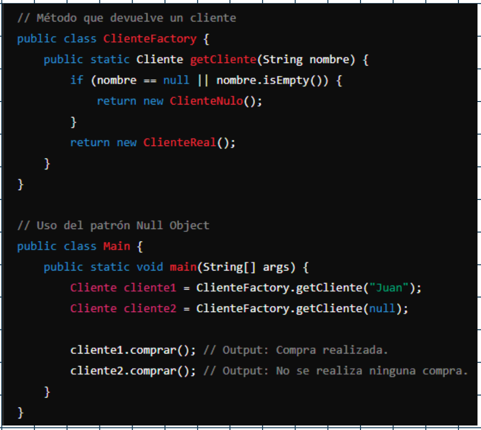
Null Object

USO: Se crea un objeto nulo en caso de que no esté lo que se busca. La idea es poder simplificar los ifs según si se devolvió null o no. Este objeto null manda mensajes de error para avisar que no se pudo hacer la acción, el problema que le encontramos es que tiene que implementar TODOS los métodos de la cl ase originar para avisar que falló

EJEMPLO:



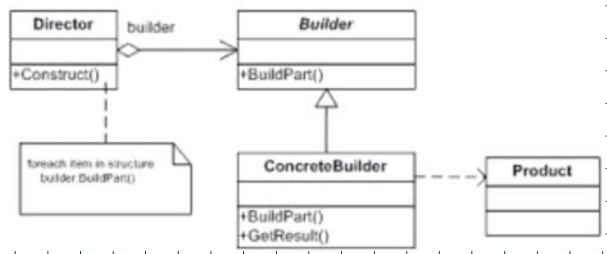
```java
public interface Cliente {
    void comprar();
}

// Implementación de un Cliente real
public class ClienteReal implements Cliente {
    @Override
    public void comprar() {
        System.out.println("Compra realizada.");
    }
}

// Implementación del Cliente Nulo
public class ClienteNulo implements Cliente {
    @Override
    public void comprar() {
        System.out.println("No se realiza ninguna compra.");
    }
}
```



```java
// Método que devuelve un cliente
public class ClienteFactory {
    public static Cliente getCliente(String nombre) {
        if (nombre == null || nombre.isEmpty()) {
            return new ClienteNulo();
        }
        return new ClienteReal();
    }
}

// Uso del patrón Null Object
public class Main {
    public static void main(String[] args) {
        Cliente cliente1 = ClienteFactory.getCliente("Juan");
        Cliente cliente2 = ClienteFactory.getCliente(null);

        cliente1.comprar(); // Output: Compra realizada.
        cliente2.comprar(); // Output: No se realiza ninguna compra.
    }
}
```

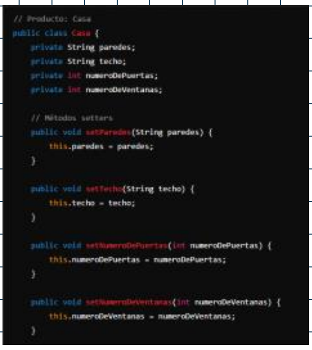
Builder

USO: es para cuando querés sacar la lógica del constructor. Haces una nueva clase cuya única responsabilidad es crear. Recibe todos los parámetros y crea un objeto mediante setters.

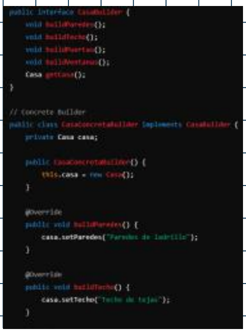


Director= (New director(new casaConcretaBuilder')

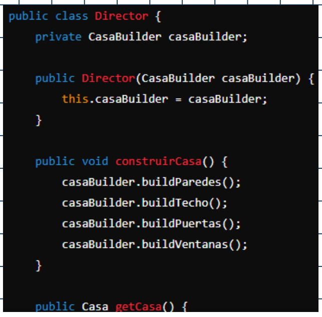
EJEMPLO: CasaBuilder es una interfaz que define qué cosas tiene una casa. CasaConcreta builder sabe hacer una casa. Después s i yo quisiera podría hacer un edificioConcreto que implemente casaConcreta y me haga un tipo de casa distinto.



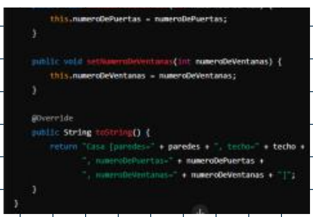
```java
// Producto: Casa
public class Casa {
    private String paredes;
    private String techo;
    private int numeroDePuertas;
    private int numeroDeVentanas;

    // Métodos setters
    public void setParedes(String paredes) {
        this.paredes = paredes;
    }

    public void setTecho(String techo) {
        this.techo = techo;
    }

    public void setNumeroDePuertas(int numeroDePuertas) {
        this.numeroDePuertas = numeroDePuertas;
    }

    public void setNumeroDeVentanas(int numeroDeVentanas) {
        this.numeroDeVentanas = numeroDeVentanas;
    }
```



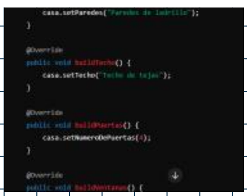
```java
public interface CasaBuilder {
    void buildParedes();
    void buildTecho();
    void buildPuertas();
    void buildVentanas();
    Casa getCasa();
}

// Concrete Builder
public class CasaConcretaBuilder implements CasaBuilder {
    private Casa casa;

    public CasaConcretaBuilder() {
        this.casa = new Casa();
    }

    @Override
    public void buildParedes() {
        casa.setParedes("Paredes de ladrillo");
    }

    @Override
    public void buildTecho() {
        casa.setTecho("Techo de tejas");
    }
```



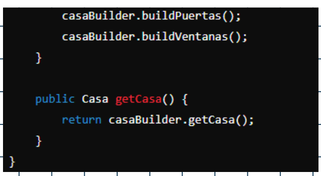
```java
public class Director {
    private CasaBuilder casaBuilder;

    public Director(CasaBuilder casaBuilder) {
        this.casaBuilder = casaBuilder;
    }

    public void construirCasa() {
        casaBuilder.buildParedes();
        casaBuilder.buildTecho();
        casaBuilder.buildPuertas();
        casaBuilder.buildVentanas();
    }

    public Casa getCasa() {
```



```java
        this.numeroDePuertas = numeroDePuertas;
    }

    public void setNumeroDeVentanas(int numeroDeVentanas) {
        this.numeroDeVentanas = numeroDeVentanas;
    }

    @Override
    public String toString() {
        return "Casa [paredes=" + paredes + ", techo=" + techo +
                ", numeroDePuertas=" + numeroDePuertas +
                ", numeroDeVentanas=" + numeroDeVentanas + "]";
    }
}
```



```java
        casa.setParedes("Paredes de ladrillo");
    }

    @Override
    public void buildTecho() {
        casa.setTecho("Techo de tejas");
    }

    @Override
    public void buildPuertas() {
        casa.setNumeroDePuertas(4);
    }

    @Override
    public void buildVentanas() {
```

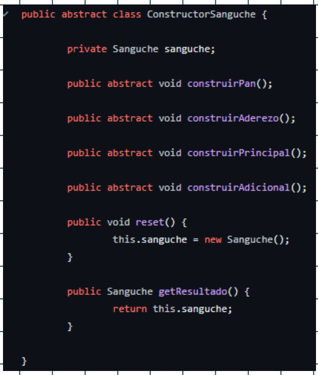


```java
        casaBuilder.buildPuertas();
        casaBuilder.buildVentanas();
    }

    public Casa getCasa() {
        return casaBuilder.getCasa();
    }
}
```

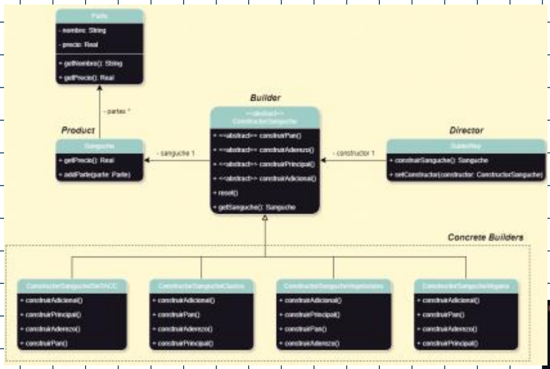
Builder

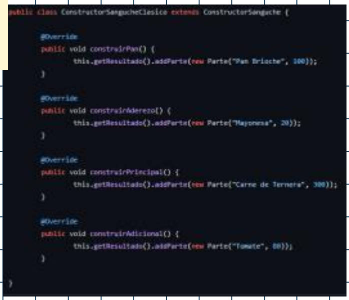
Es medio parecido al factory en sentido de que ambos son creacionales pero la diferencia está en que factory como que cambia la lógica con la que crea cosas mientras que builder es simplemente para crear el objeto.
Siguiendo con el ejemplo del juego que usamos para el factory, ese patrón cambia la lógica con la que se crean enemigos mient ras que este lo que haría, por ejemplo, es ocuparse de crear goombas azules y rojos. Tendría dos objetos con estructura parecida pero distintos valores y un concreteGoombaBuilder q ue se encargue de crearlos con estos valores diferentes.

Grabación de audio iniciada: 23:54 lunes, 27 de mayo de 2024
En builder las cosas tienen las mismas variables, en factory no necesariamente.

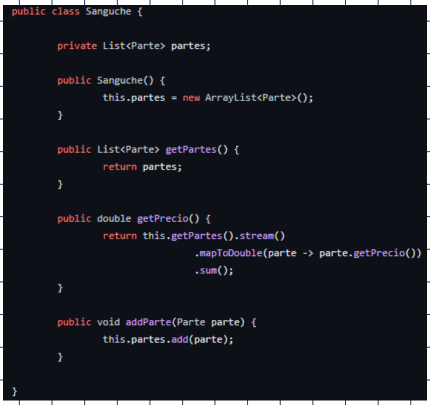


```java
public abstract class ConstructorSanguche {

        private Sanguche sanguche;

        public abstract void construirPan();

        public abstract void construirAderezo();

        public abstract void construirPrincipal();

        public abstract void construirAdicional();

        public void reset() {
                this.sanguche = new Sanguche();
        }

        public Sanguche getResultado() {
                return this.sanguche;
        }

}
```

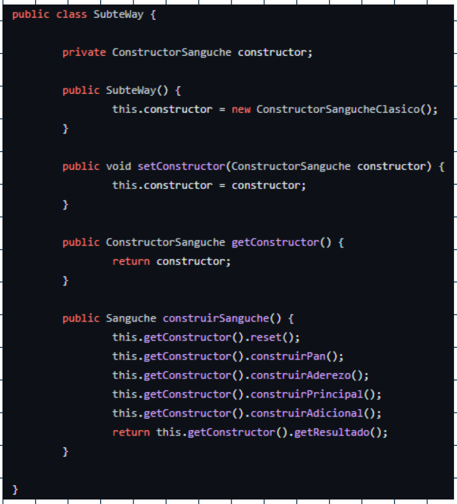




```java
public class ConstructorSangucheClasico extends ConstructorSanguche {

        @Override
        public void construirPan() {
                this.getResultado().addParte(new Parte("Pan Brioche", 100));
        }

        @Override
        public void construirAderezo() {
                this.getResultado().addParte(new Parte("Mayonesa", 20));
        }

        @Override
        public void construirPrincipal() {
                this.getResultado().addParte(new Parte("Carne de Ternera", 300));
        }

        @Override
        public void construirAdicional() {
                this.getResultado().addParte(new Parte("Tomate", 80));
        }

}
```

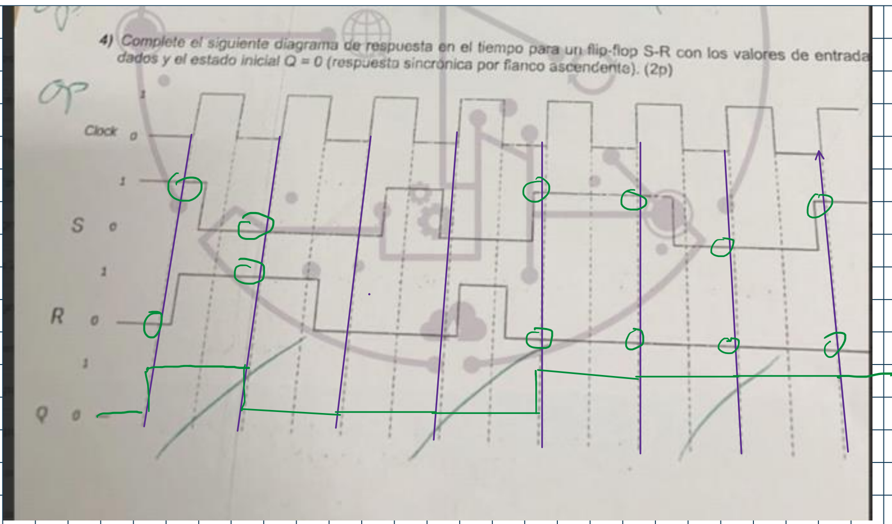


```java
public class Sanguche {

        private List<Parte> partes;

        public Sanguche() {
                this.partes = new ArrayList<Parte>();
        }

        public List<Parte> getPartes() {
                return partes;
        }

        public double getPrecio() {
                return this.getPartes().stream()
                                .mapToDouble(parte -> parte.getPrecio())
                                .sum();
        }

        public void addParte(Parte parte) {
                this.partes.add(parte);
        }

}
```



```java
public class SubteWay {

        private ConstructorSanguche constructor;

        public SubteWay() {
                this.constructor = new ConstructorSangucheClasico();
        }

        public void setConstructor(ConstructorSanguche constructor) {
                this.constructor = constructor;
        }

        public ConstructorSanguche getConstructor() {
                return constructor;
        }

        public Sanguche construirSanguche() {
                this.getConstructor().reset();
                this.getConstructor().construirPan();
                this.getConstructor().construirAderezo();
                this.getConstructor().construirPrincipal();
                this.getConstructor().construirAdicional();
                return this.getConstructor().getResultado();
        }

}
```

4) *Complete el siguiente diagrama de respuesta en el tiempo para un flip-flop S-R con los valores de entrada dados y el estado inicial Q = 0 (respuesta sincrónica por flanco ascendente). (2p)*



3) Indique la ecuación F que se corresponde con la siguiente tabla de verdad

F= $(\bar{A}.B.\bar{C})+(\bar{A}.B.C)+(A.\bar{B}.\bar{C})$ (2p)

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

$(\bar{A}.B).(\bar{C}+C)+(A.\bar{B}.\bar{C})$

$\rightarrow (\bar{A}.B)+(A.\bar{B}.\bar{C})$

El siguiente programa recorre una tabla que comienza en la dirección de memoria TABLA y guarda en la dirección PARES la cantidad de números pares que contiene la tabla.

```
1                  ORG 1000h
2  RELLENO          DW  0
3  TABLA            DB  10, 7, 14, 23, 49, 50
4  FIN_TABLA        DB  ?
5  PARES            DB  ?
6
7                   ORG 2000h
8                   ; instrucción a completar
9                   MOV DL, OFFSET FIN_TABLA - OFFSET TABLA
10          BUCLE:  MOV AL, [BX]
11                  AND AL, 1
12                  JNZ IMPAR
13                  INC CL
14          IMPAR:  INC BX
15                  DEC DL
16                  JNZ BUCLE
17                  ; instrucción a completar
18                  HLT
19                  END
```

5) ¿Qué modo de direccionamiento utiliza el segundo operando de la instrucción de la línea 10?

[redacted]____ (1p)

6) ¿Cómo debería completarse la línea 8?

[redacted] (2p)

7) ¿Cuántas veces se toma el salto de la instrucción JNZ BUCLE (línea 16)?

Cantidad: [redacted]____ (2p)

8) ¿Cómo debería completarse la línea 17?

[redacted]____ (2p)

9) ¿En qué dirección de memoria está almacenado el valor 23?

Dir: [redacted]____ (1p)

El siguiente programa cuenta cuantas números CIFRA hay en NUMEROS y almacena el resultado final en RESUL

```
1.                ORG 1000H
2.  NUMEROS       DB    2,1,3,5,3,2,3,7,3,8
3.  FINTABLA      DB     0
4.  CIFRA         DB    3
5.  RESUL         DB     0
6.
7.                ORG 3000H
8.  SUB1:         MOV AL, [BX]
9.                INC BX
10.               CMP AL, CIFRA ←
11.               JNZ OTRO
12. SUMAR:        INC RESUL
13. OTRO:         DEC CL
14.               JZ FIN
15.               JMP SUB1
16. FIN:          ____________
17.
18.               ORG 2000H
19.               MOV BX, OFFSET NUMEROS
20.               MOV CL, ________
21.               CALL SUB1
22.               HLT
23.               END
```

3) ¿Qué Modo de direccionamiento utiliza la instrucción de la línea 10 ?

__________ (1p)

4) ¿Cómo debería completarse la línea 16?

__________ (2p)

5) ¿Cuántas veces se ejecuta la instrucción (línea 9) INC BX?

Cantidad: __________ (2p)

6) ¿Cómo debería completarse la línea 20?

__________ (2p)

7) ¿En qué dirección de memoria está almacenado el valor 3 de CIFRA?

Dir: __________ (2p)

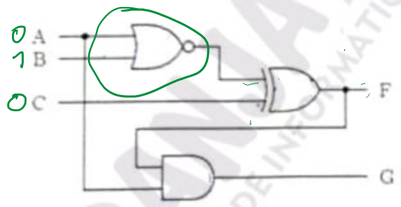
8) Al finalizar la ejecución del programa dado, ¿qué valor queda guardado en RESUL?

__________ (1p)

9) Al finalizar el programa, ¿qué valor queda almacenado en el registro BX?

BX = __________ (2p)

5) Dado el siguiente circuito, complete la tabla de verdad con la salida de la función F.



A	B	C	F	
0	0	0		(0, 25p)
0	0	1		(0, 25p)
0	1	0		(0, 25p)
0	1	1		(0, 25p)
1	0	0		(0, 25p)
1	0	1		(0, 25p)
1	1	0		(0, 25p)
1	1	1		(0, 25p)

A	B	C	A nor b=j	J xor c
0	0	0	1	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	0
1	1	1	0	1

si A = 1, B = 1 y C = 0: ¿Cuál es el valor de la salida G? G= 0 (1p)

6) Escriba las ecuaciones que relacionan las entradas del circuito del punto 5 con las salidas del mismo

F = $\overline{A+B} \oplus C$ (1, 5p)

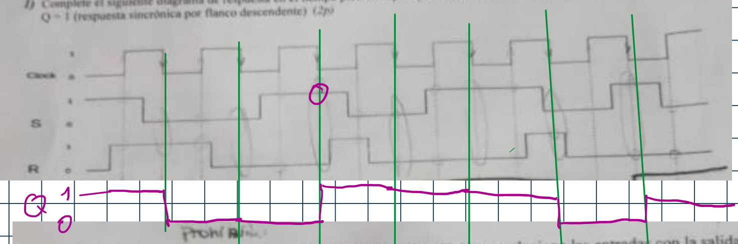
G = $\left(\overline{A+B} \oplus C\right) \cdot A$ (1, 5p)

1) Dado un byte **X**, indique en la columna de la izquierda las operaciones lógicas junto con sus máscaras para **poner en uno** los bits 0 y 7, **poner en cero** los bits 3 y 6 e **invertir** los bits 1 y 5, dejando inalterados al resto de los bits (no use más de tres operaciones lógicas para lograrlo). Dado otro byte **Y**, escriba en la columna de la derecha los resultados de aplicar las operaciones lógicas indicadas.

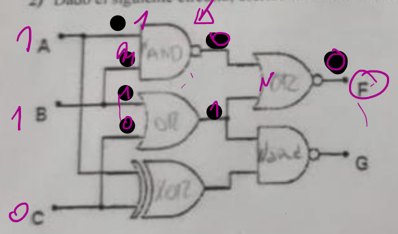
```
           xxxxxxxx                                  yyyyyyyy
 ----------------------- (0,5p)                 NAND 01100101
           ------------                          ------------ (0,5p)
 ----------------------- (0,5p)                 XNOR 11001100
           ------------                          ------------ (0,5p)
 ----------------------- (0,5p)                 NOR  10100110
           10x̄x0xx̄1                              ------------ (0,5p)
```

Observaciones: Escribir respuestas con tinta. Cada ejercicio …
MÁXIMA 14.

1) Complete el siguiente diagrama de respuesta en el tiempo para un flip-flop S-R con los valores de entrada dados y el estado inicial Q = 1 (respuesta sincrónica por flanco descendente) (2p)



2) Dado el siguiente circuito, escriba la ecuación (función) que relaciona las entradas con la salida G del mismo.



G $\overline{(B+C)\cdot(A\oplus C)}$

3) ¿Cuál será el valor de la salida F si A =1, B = 1 y C = 0?

F= [redacted] (1p)

5) Si AX=228, CX=152, DX=200. Que valores quedan almacenados en AX, CX y DX tras ejecutar la siguiente secuencia de instrucciones:

```
PUSH DX
PUSH CX
POP  DX
PUSH AX
POP  CX
POP  AX
```

AX = [redacted] (1p)

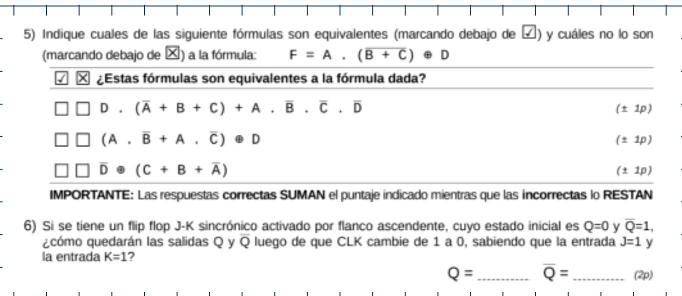
DX = [redacted] (1p)

4) ¿Cuál debe ser la última instrucción de una subrutina ?

[redacted] (1p)

1) Dado un byte **X**, indique en la columna de la izquierda las operaciones lógicas junto con sus máscaras para **poner en uno** los bits 0 y 7, **poner en cero** los bits 3 y 6 e **invertir** los bits 1 y 5, dejando inalterados al resto de los bits (no use más de tres operaciones lógicas para lograrlo). Dado otro byte **Y**, escriba en la columna de la derecha los resultados de aplicar las operaciones lógicas indicadas.

```
           xxxxxxxx                                  yyyyyyyy
 ----------------------- (0,5p)                 NAND 01100101
           ------------                          ------------ (0,5p)
 ----------------------- (0,5p)                 XNOR 11001100
           ------------                          ------------ (0,5p)
 ----------------------- (0,5p)                 NOR  10100110
           10x̄x0xx̄1                              ------------ (0,5p)
```



5) Indique cuales de las siguiente fórmulas son equivalentes (marcando debajo de ☑) y cuáles no lo son (marcando debajo de ☒) a la fórmula: $F = A \,.\, (\overline{B + C}) \oplus D$

☑	☒	**¿Estas fórmulas son equivalentes a la fórmula dada?**	
☐	☐	$D \,.\, (\bar{A} + B + C) + A \,.\, \bar{B} \,.\, \bar{C} \,.\, \bar{D}$	*(± 1p)*
☐	☐	$(A \,.\, \bar{B} + A \,.\, \bar{C}) \oplus D$	*(± 1p)*
☐	☐	$\bar{D} \oplus (C + B + \bar{A})$	*(± 1p)*

IMPORTANTE: Las respuestas **correctas SUMAN** el puntaje indicado mientras que las **incorrectas** lo **RESTAN**

6) Si se tiene un flip flop J-K sincrónico activado por flanco ascendente, cuyo estado inicial es Q=0 y $\bar{Q}$=1, ¿cómo quedarán las salidas Q y $\bar{Q}$ luego de que CLK cambie de 1 a 0, sabiendo que la entrada J=1 y la entrada K=1?

Q = ... $\bar{Q}$ = ... *(2p)*