# Patrones de diseño

# STRATEGY

Marina Donis - Paradigmas VI 2021

# INTRODUCCIÓN - EJEMPLO

- Se quiere implementar un videojuego en donde el *personaje* protagonista puede pelear con otros dos personajes para así obtener sus *armas*.
- Para pelear cada uno de los personajes utiliza su *arma* particular. Una vez que el protagonista le gana a otro personaje, obtiene su *arma* y puede utilizarla en la próxima pelea.
- Para evadir ataques los personajes *saltan*. Por defecto todos los personajes pueden realizar un salto corto. El protagonista puedn elegir realizar un salto largo o salto corto.

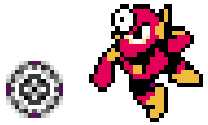
# PERSONAJES Y COMPORTAMIENTOS

## MegaMan 

- El **protagonista** del juego.
- Su ***arma*** por defecto es el **Mega Buster**.
- Puede usar las ***armas*** de los demás personajes si les gana en una pelea.
- Puede realizar los **dos *saltos***.

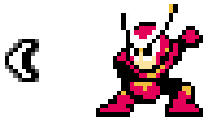
## MetalMan 

- Uno de los **personajes** con los que pelea MegaMan.
- Su ***arma*** por defecto es la **Metal Blade**.
- Puede realizar un ***salto*** **corto**.



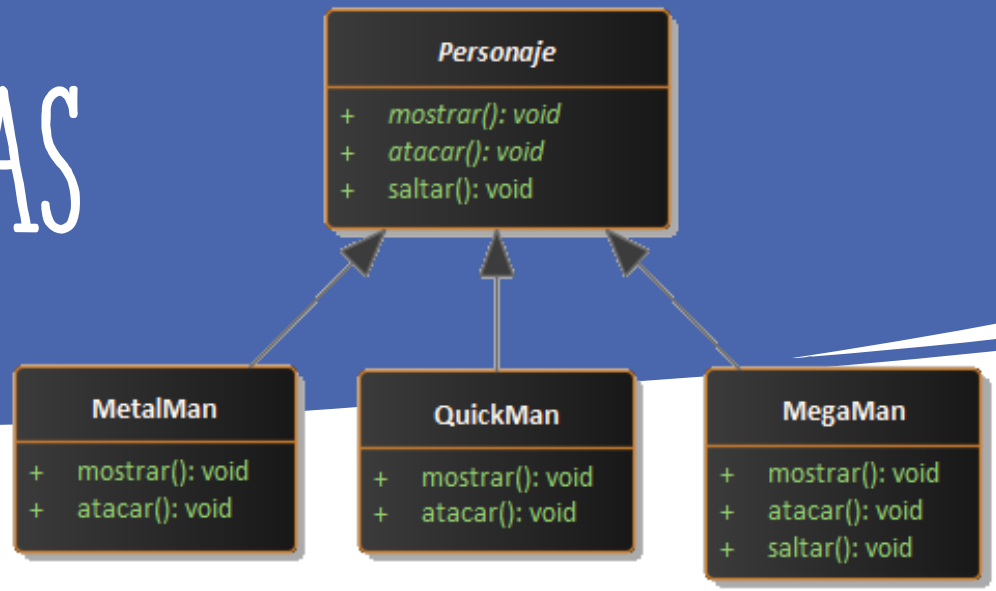
## QuickMan 

- Uno de los **personajes** con los que pelea MegaMan.
- Su ***arma*** por defecto es el **Quick Boomerang**.
- Puede realizar un ***salto*** **corto**.



# PROBLEMAS

- Los comportamientos de MetalMan y QuickMan son fijos, pero el protagonista debe cambiar de arma y tipo de salto en tiempo de ejecución
- ¿Qué pasa si se agrega otro personaje que pueda usar todas las armas pero sólo pueda realizar un salto largo?



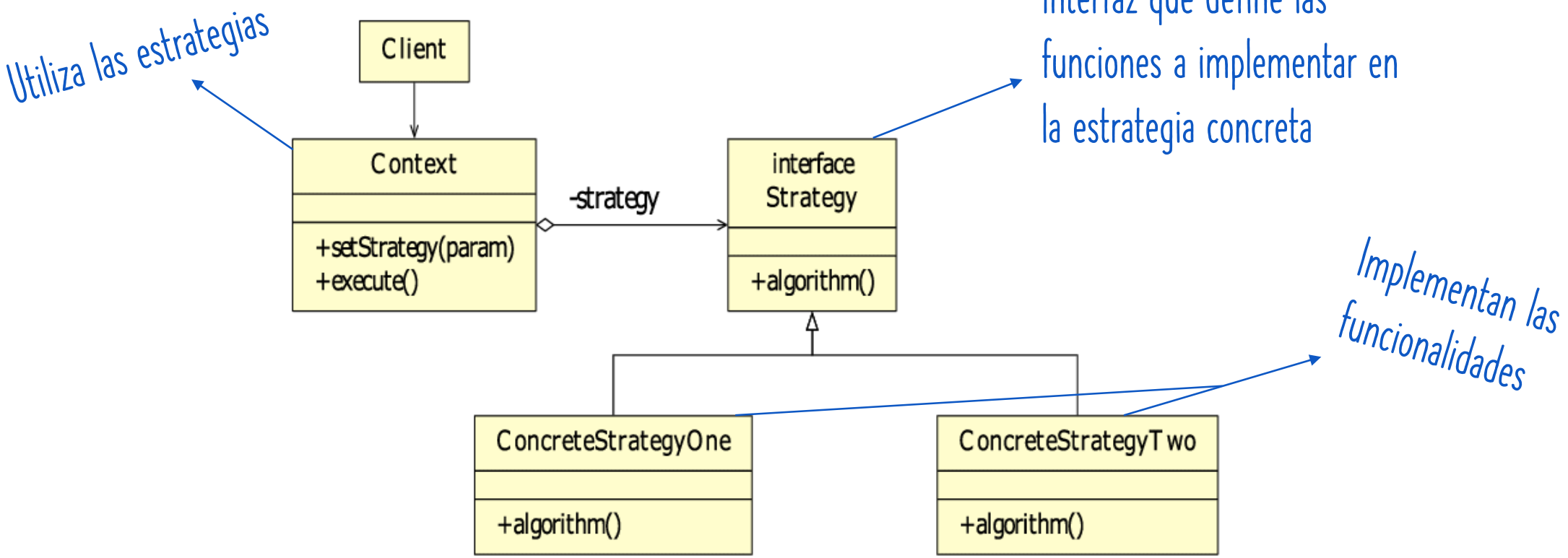

- ¿Qué pasa si se quiere agregar otros personajes que peleen con MegaMan?
- Cada uno tendrá definido un tipo de salto y un arma nueva que tenemos que agregar
- Tenemos que cambiar el comportamiento atacar() de MegaMan para que pueda usar las nuevas armas

- Definimos el comportamiento saltar() en la clase base, pero hay que reescribirlo para quienes usen otro tipo de salto o no puedan saltar
- Complicado agregarle funcionalidades nuevas y difícil de mantener

# EL PATRÓN - STRATEGY

- Toma lo **variable** y lo **encapsula** para que no afecte el resto del código.
- Define una **familia de algoritmos**, los encapsula y los vuelve intercambiables.
- La clase base **delega sus comportamientos** a otra clase que **sólo se encarga de definir la implementación del comportamiento**.
- Permite que el algoritmo varíe **independientemente de quienes lo utilizan** por lo que es más simple de mantener.

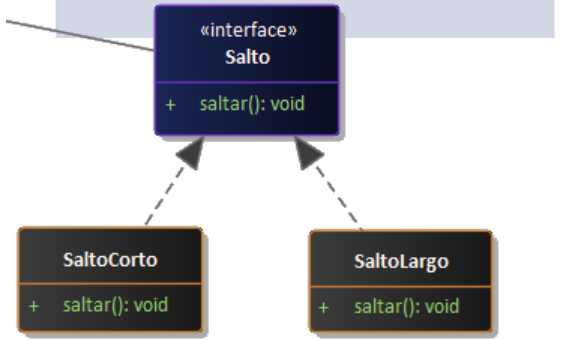
# CONCEPTO


Utiliza las estrategias
Client
Context
+setStrategy(param)
+execute()
-strategy
interface
Strategy
+algorithm()
Interfaz que define las funciones a implementar en la estrategia concreta
Implementan las funcionalidades
ConcreteStrategyOne
+algorithm()
ConcreteStrategyTwo
+algorithm()

# CÓMO APLICARLO

1

Separamos Arma y Salto de personaje creando interfaces y clases que implementen los comportamientos




```
public class SaltoCorto implements Salto {
    public void saltar() {
        System.out.println("Realizo un salto corto");
    }
}
```

2

En la clase base, definimos **variables que representen los comportamientos**


```
public abstract class Personaje {
    Salto salto;
    Arma arma;
```

3

Cambiamos los métodos que definían el comportamiento para que deleguen la implementación a los objetos que referencian las variables

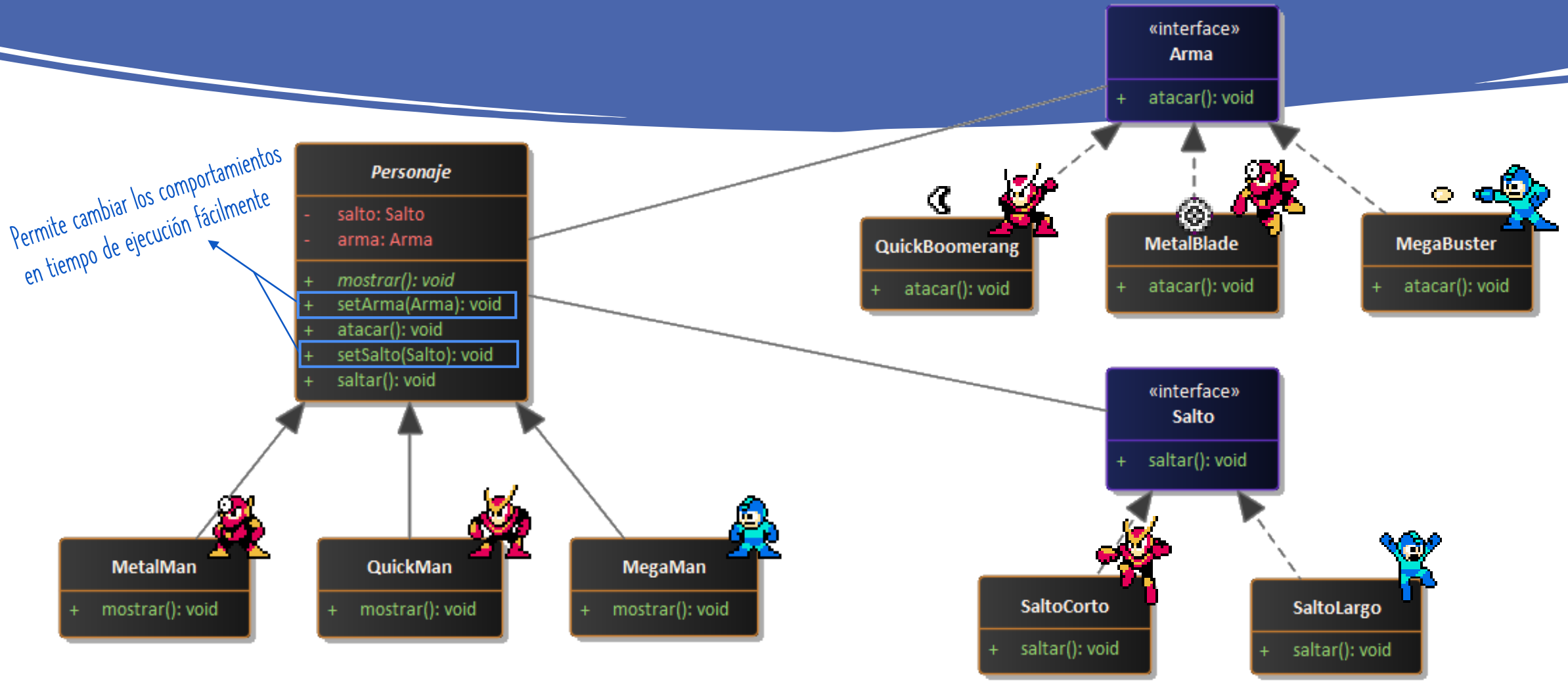

```
public void saltar() { salto.saltar(); }

public void atacar() { arma.atacar(); }
```

4

Cuando creamos una instancia de la subclase, le asignamos un objeto que corresponda al comportamiento que queremos en las variables

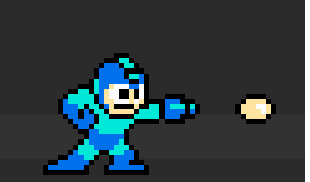
```
public class MetalMan extends Personaje {
    public MetalMan() {
        salto = new SaltoCorto();
        arma = new MetalBlade();
    }
```

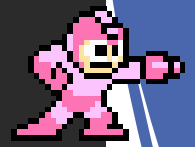
# DELEGANDO COMPORTAMIENTOS

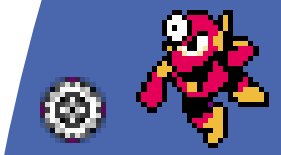


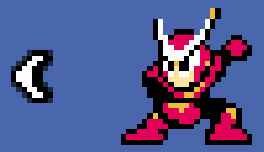
# IMPLEMENTACIÓN Y PRUEBA

```java
public static void main(String[] args) {
    Personaje metalMan = new MetalMan();
    Personaje quickMan = new QuickMan();
    Personaje megaMan = new MegaMan();

    megaMan.mostrar();
    megaMan.saltar();
    megaMan.atacar();

    metalMan.mostrar();
    metalMan.saltar();
    metalMan.atacar();

    megaMan.setArma(new MetalBlade());  // obtiene y cambia de arma
    megaMan.mostrar();
    megaMan.saltar();
    megaMan.atacar();
    megaMan.setSalto(new SaltoLargo());  // cambia de salto
    megaMan.saltar();

    quickMan.mostrar();
    quickMan.saltar();
    quickMan.atacar();

    megaMan.setArma(new QuickBoomerang());  // obtiene y cambia de arma
    megaMan.mostrar();
    megaMan.atacar();
```









```
Soy megaman
Realizo un salto corto
Ataco con el mega buster
Soy metal man
Realizo un salto corto
Ataco con la metal blade
Soy megaman
Realizo un salto corto
Ataco con la metal blade
Realizo un salto largo
Soy quick man
Realizo un salto corto
Ataco con el quick boomerang
Soy megaman
Ataco con el quick boomerang

Process finished with exit code 0
```