**TECNOLÓGICO NACIONAL DE MÉXICO**
**INSTITUTO TECNOLÓGICO DE TIJUANA**

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

Tarea

# Patrón template

**Docente**
José de Jesús Parra Galaviz

SEMESTRE agosto – diciembre 2021

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES

Patrones de diseño

Sergio Alberto Garza Aguilar 15211700

10 de septiembre del 2021

## Estructura

Este patrón es de tipo comportamiento y define un esqueleto algorítmico permitiendo así que la superclase pueda alojar métodos abstractos para luego ser redefinidos por las subclases sin la necesidad de tener que modificar la estructura base.

### Características

- Las subclases pueden redefinir ciertas partes de la clase base sin alterar la estructura básica.
- Las superclases pueden delegar tareas a las subclases para que implementen la funcionalidad en su propia clase.

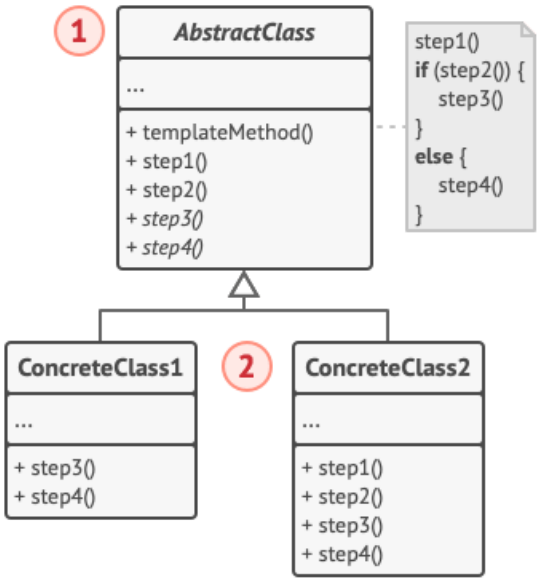
### ¿En qué casos se recomienda utilizarlo?

- Cuando detectes pasos que tendrán que repetirse una y otra vez.
- Cuando tengas ansiedad adictiva a la instrucción **Switch**.
- Cuando desconozcas la magnitud que pueda tener una superclase para las subclases.

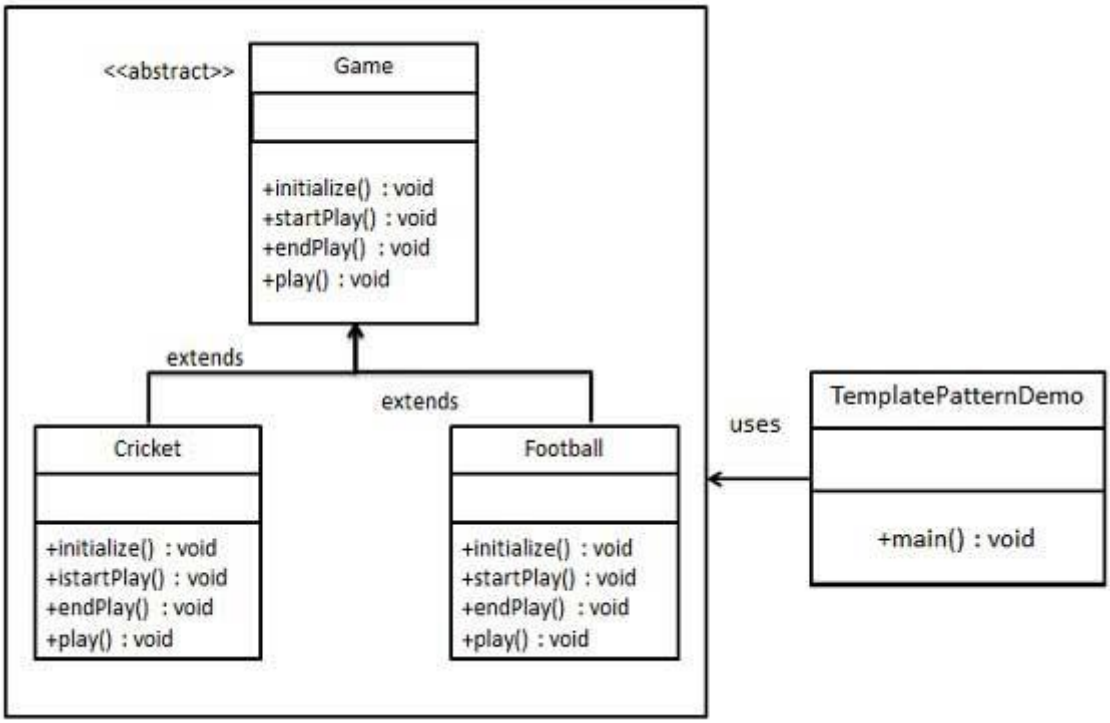
### ¿Cuál es el beneficio de aplicar este patrón?

- Se puede reutilizar una clase abstracta en las subclases
- No se modifica el comportamiento de la clase Base.
- Nada afecta a la superclase porque la herencia solo extiende la funcionalidad.

## Modelo del patrón



## Ejemplo de uso




Vamos a crear una clase abstracta de Game *que* defina operaciones con un método de plantilla configurado como final para que no se pueda anular. El Criket y el Football on clases concretas que amplían el *Game* y anulan sus métodos.
*TemplatePatternDemo* , nuestra clase de demostración, usará *Game* para demostrar el uso del patrón de plantilla.

1.- Cree una clase abstracta con un método de plantilla que sea definitivo.

```
public abstract class Game {
   abstract void initialize();
   abstract void startPlay();
   abstract void endPlay();

   //template method
   public final void play(){

      //initialize the game
      initialize();

      //start game
      startPlay();

      //end game
      endPlay();
   }
}
```

2.- Cree clases concretas ampliando la clase anterior.

*Cricket.java*

```
public class Cricket extends Game {

   @Override
   void endPlay() {
      System.out.println("Cricket Game Finished!");
   }

   @Override
   void initialize() {
      System.out.println("Cricket Game Initialized! Start playing.");
   }

   @Override
   void startPlay() {
      System.out.println("Cricket Game Started. Enjoy the game!");
   }
}
```

*Football.java*

```
public class Football extends Game {

   @Override
   void endPlay() {
      System.out.println("Football Game Finished!");
   }

   @Override
   void initialize() {
      System.out.println("Football Game Initialized! Start playing.");
   }

   @Override
   void startPlay() {
      System.out.println("Football Game Started. Enjoy the game!");
   }
}
```

3.- Utilice *Game*'s template method play() para demostrar de una manera definida de jugar el juego.

*TemplatePatternDemo.java*

```
public class TemplatePatternDemo {
   public static void main(String[] args) {

      Game game = new Cricket();
      game.play();
      System.out.println();
      game = new Football();
      game.play();
   }
}
```

4.- Verifique la salida.

```
Cricket Game Initialized! Start playing.
Cricket Game Started. Enjoy the game!
Cricket Game Finished!

Football Game Initialized! Start playing.
Football Game Started. Enjoy the game!
Football Game Finished!
```

## Bibliografía


- Tutorials Point (2021), Desing patterns, Template Pattern (https://www.tutorialspoint.com/design_pattern/template_pattern.htm)

- Carlos Caballero (13 abril del 2020), Patrones de diseño, Template Method, Un patrón de diseño explicado con Pokémon. (https://medium.com/dottech/patrones-de-dise%C3%B1o-template-method-1392e0a5dc74)