Programas de Diseño de Patrones

Singleton

- Descripción del patrón Singleton
  - El patrón Singleton asegura que una clase tenga una única instancia y proporciona un punto de acceso global a ella. Es útil para gestionar recursos compartidos.
- Ejemplo de implementación en Python

```
class Singleton:
_instance = None

def new(cls):
if cls._instance is None:
cls._instance = super(Singleton, cls).new(cls)
return cls._instance
```

- Contexto de uso
  - Se utiliza en configuraciones de aplicaciones donde se necesita un único objeto, como un gestor de configuración.

Factory Method

- Descripción del patrón Factory Method
  - El patrón Factory Method permite crear objetos sin especificar la clase exacta del objeto que se va a crear. Facilita la creación de familias de objetos.
- Ejemplo de implementación en Python

```
class Product:
def operation(self):
pass

class ConcreteProduct(Product):
def operation(self):
return "Resultado del producto concreto."

class Creator:
def factory_method(self):
return ConcreteProduct()
```

- Contexto de uso
  - Ideal para sistemas que deben ser extensibles y donde se pueden agregar nuevos productos sin modificar el código existente.

Builder

- Descripción del patrón Builder
  - El patrón Builder separa la construcción de un objeto complejo de su representación, permitiendo crear diferentes representaciones del mismo tipo de objeto.
- Ejemplo de implementación en Python

```
class Product:
def init(self):
self.parts = []

class Builder:
def build_part(self):
pass

class ConcreteBuilder(Builder):
def build_part(self):
self.product.parts.append("Parte construida")
```

- Contexto de uso
  - Útil en la construcción de objetos complejos, como vehículos o casas, donde se requiere un proceso de construcción paso a paso.

Abstract Factory

- Descripción del patrón Abstract Factory
  - El patrón Abstract Factory proporciona una interfaz para crear familias de objetos relacionados sin especificar sus clases concretas. Promueve la independencia de la implementación.
- Ejemplo de implementación en Python

```
class AbstractFactory:
def create_product_a(self):
pass

def create_product_b(self):
pass

class ConcreteFactory1(AbstractFactory):
def create_product_a(self):
return ProductA1()

def create_product_b(self):
return ProductB1()
```

- Contexto de uso
  - Se utiliza en aplicaciones que requieren la creación de múltiples productos relacionados, como interfaces gráficas que pueden cambiar de apariencia.

Comparación de Patrones

- Diferencias clave entre los patrones
  - Singleton vs Factory Method
    - El Singleton se centra en la creación de una única instancia, mientras que el Factory Method se enfoca en la creación de objetos sin especificar su clase.
  - Abstract Factory vs Builder
    - El Abstract Factory crea familias de objetos relacionados, mientras que el Builder se ocupa de la construcción de un objeto complejo a través de un proceso gradual.
  - Cada patrón tiene un propósito y contexto de uso diferente, lo que los hace adecuados para diversas situaciones en el desarrollo de software.