Los tipos de patrones se refieren a las diferentes categorías en las que se pueden clasificar los patrones en general. Algunas de las clasificaciones comunes son:

Patrones de diseño: Son soluciones probadas y documentadas para problemas comunes en el diseño de software. Estos patrones proporcionan un enfoque estándar y reutilizable para resolver problemas de diseño específicos.

Patrones arquitectónicos: Son patrones que se utilizan para diseñar la estructura y el comportamiento de sistemas de software a gran escala. Estos patrones abordan desafíos arquitectónicos y proporcionan una guía para organizar y relacionar componentes del sistema.

Patrones de creación: Se centran en el proceso de creación de objetos y abordan cómo se crean y se instancian las instancias de clases. Estos patrones se utilizan para garantizar que la creación de objetos sea flexible, reutilizable y siga principios de diseño sólidos.

Patrones de comportamiento: Se centran en la interacción y comunicación entre objetos, y cómo se distribuyen las responsabilidades entre ellos. Estos patrones ayudan a definir algoritmos y comportamientos complejos dentro de un sistema de software.

Patrones de optimización: Son patrones que se utilizan para mejorar el rendimiento y la eficiencia de los sistemas de software existentes. Estos patrones ofrecen técnicas y enfoques para optimizar el código y mejorar los recursos utilizados.

Un patrón de software es una solución general y reutilizable para un problema común en el diseño o desarrollo de software. Los patrones de software proporcionan un enfoque probado y documentado para resolver problemas específicos, basado en la experiencia acumulada de la comunidad de desarrollo de software.

Los patrones de software se pueden clasificar en varias categorías, según su propósito y área de aplicación. Algunas de las clasificaciones comunes son:

Patrones de creación: Se utilizan para la creación de objetos y abordan cómo se crean las instancias de clases.

Patrones de estructura: Se centran en la composición y organización de clases y objetos para formar estructuras más grandes.

Patrones de comportamiento: Se centran en la interacción y comunicación entre objetos y cómo se distribuyen las responsabilidades.

Los patrones de creación se centran en el proceso de creación de objetos. Algunos patrones de creación comunes incluyen:

Patrón de fábrica (Factory): Proporciona una interfaz para crear objetos, pero permite que las subclases decidan qué clase instanciar. Permite la creación de objetos sin especificar la clase exacta.

Patrón de constructor (Builder): Se utiliza para construir objetos complejos paso a paso. Permite la creación de diferentes representaciones de un objeto utilizando el mismo proceso de construcción.

Patrón de prototipo (Prototype): Permite la creación de nuevos objetos a partir de un objeto existente mediante la clonación. Evita la creación de subclases para crear objetos nuevos.

Patrón de Singleton: Garantiza que solo haya una instancia de una clase en todo el programa. Proporciona un punto de acceso global a esa instancia.

Ejemplos de uso de patrones de creación:

Patrón de fábrica: Imagina un sistema de una aplicación de comercio electrónico que debe manejar diferentes tipos de productos. En lugar de tener un código que cree objetos de productos directamente, se puede utilizar un patrón de fábrica para encapsular la lógica de creación y tener una interfaz común para crear diferentes tipos de productos.

Patrón de constructor: Supongamos que estás construyendo un juego de video y tienes un personaje con diferentes atributos, como nombre, fuerza y velocidad. En lugar de tener un constructor con muchos parámetros, puedes utilizar un patrón de constructor para crear un objeto de personaje paso a paso, estableciendo los atributos relevantes según sea necesario.

Patrón de prototipo: Supongamos que tienes un editor de gráficos y quieres permitir a los usuarios duplicar elementos existentes. Puedes utilizar el patrón de prototipo para clonar objetos existentes y crear nuevos objetos basados en ellos, evitando la necesidad de crear subclases para cada tipo de objeto.

Patrón de Singleton: En un sistema de registro, puedes utilizar el patrón de Singleton para asegurarte de que solo haya una instancia del registro en todo el programa. Esto garantiza que todos los componentes del sistema accedan al mismo registro y evita instancias múltiples y problemas de coherencia en el registro.