**What is the Factory Method Design Pattern?**

The Factory Method Design Pattern is a creational design pattern used in software engineering to provide an interface for creating objects in a superclass, while allowing subclasses to alter the type of objects that will be created. It encapsulates the object creation logic in a separate method, abstracting the instantiation process and promoting loose coupling between the creator and the created objects. This pattern enables flexibility, extensibility, and maintainability in the codebase by allowing subclasses to define their own implementation of the factory method to create specific types of objects.

**When to use Factory Method Design Pattern?**

Use Factory Method Design Pattern:

* **When you want to encapsulate object creation:** If you have a complex object creation process or if the process may vary based on conditions, encapsulating this logic in a factory method can simplify client code and promote reusability.
* **When you want to decouple client code from concrete classes:** Using the Factory Method Pattern allows you to create objects through an interface or abstract class, abstracting away the specific implementation details of the concrete classes from the client code. This promotes loose coupling and makes it easier to modify or extend the system without impacting existing client code.
* **When you need to support multiple product variations:** If your application needs to create different variations of a product or if new types of products may be introduced in the future, the Factory Method Pattern provides a flexible way to accommodate these variations by defining factory methods for each product type.
* **When you want to support customization or configuration:** Factories can be used to encapsulate configuration logic, allowing clients to customize the creation process by providing parameters or configuration options to the factory method.

**Advantages of Factory Method Design Pattern**

The advantages of Factory Method Design Pattern are:

* **Decoupling:** It separates object creation logic from the client code that uses those objects. This makes the code more flexible and maintainable because changes to the creation process don’t require modifications to client code.
* **Extensibility:** It’s easy to introduce new product types without changing the client code. You simply need to create a new Concrete Creator subclass and implement the factory method to produce the new product.
* **Testability:** It simplifies unit testing by allowing you to mock or stub out product creation during tests. You can test different product implementations in isolation without relying on actual object creation.
* **Code Reusability:** The factory method can be reused in different parts of the application where object creation is needed. This promotes centralizing and reusing object creation logic.
* **Encapsulation:** It hides the concrete product classes from the client code, making the code less dependent on specific implementations. This improves maintainability and reduces coupling.

**Disadvantages of Factory Method Design Pattern**

The disavantages of Factory Method Design Pattern are:

* **Increased Complexity:** It introduces additional classes and interfaces, adding a layer of abstraction that can make the code more complex to understand and maintain, especially for those unfamiliar with the pattern.
* **Overhead:** The use of polymorphism and dynamic binding can slightly impact performance, although this is often negligible in most applications.
* **Tight Coupling Within Product Hierarchies:** Concrete Creators are still tightly coupled to their corresponding Concrete Products. Changes to one often necessitate changes to the other.
* **Dependency on Concrete Subclasses:** The client code still depends on the abstract Creator class, requiring knowledge of its concrete subclasses to make correct factory method calls.
* **Potential for Overuse:** It’s important to use the Factory Method pattern judiciously to avoid over-engineering the application. Simple object creation can often be handled directly without the need for a factory.
* **Testing Challenges:** Testing the factory logic itself can be more complex.