Factory design pattern:  
**Other name (if any)**

The Factory Method Pattern is also known as **Virtual Constructor.**

**What it does**

This type of design pattern comes under creational pattern. This pattern is used when we want to return different type of objects based on some condition. The factory class create object based on the input passed to it. So client does not need to do worry about the object creation. This pattern enhances maintainability by centralizing the creation logic,

**Where to use**

1. aims for decoupling of creation logic from client code,   
2. facilitates dynamic object creation.  
3. This pattern simplifies instantiation processes,   
4. promotes code flexibility,   
5. is beneficial in scenarios where creating objects involves intricate steps or may change over time.

**Steps**

1. Interface Base Class: create an interface for base class  
2. Implemented Child Class: It will implements the base class. Here the main code will be written for that specific class.  
3. Factory class: There will be a function which will return the specific implemented child class based on the condition.  
4. Client code: It will create object using factoryClass. Pass the value to the function of the factory class. Then call the function of specific implemented child class.

**Advantages**

1. Because of factory design pattern we do not need to import too many classes in the client code.   
2. We can add different variation of classes without doing anything to the client code.  
3. Factory design pattern helps organize code better.  
4. It centralizes object creation in one place.  
5. This makes code easier to maintain and understand.  
6. Factories can hide complex instantiation logic.  
7. It enhances flexibility by allowing easy changes to object creation.  
8. Factories enable dependency injection (dependency injection is a way to pass objects (dependencies) into a class, rather than the class creating them itself)  and inversion of control (it shifts the responsibility of object creation and lifecycle management from the application code to a container or framework.).

**Disadvantages**

1. Factory design pattern can introduce complexity, especially in smaller projects where it might be overkill.  
2. It requires additional classes and interfaces, which can lead to a larger codebase.  
3. Overusing factories can make code harder to understand for developers unfamiliar with the pattern.  
4. Implementing the pattern without proper planning can result in unnecessary overhead.  
5. It might not be suitable for simple or straightforward object creation scenarios.  
6. Using factories excessively can hinder the ability to leverage language features like constructors.  
7. It may lead to a proliferation (it can refer to the uncontrolled growth or multiplication of certain elements, such as classes, methods, or components within a codebase.) of factory methods, potentially cluttering the codebase.  
8. Factories can sometimes obscure the direct relationship between components, making the code harder to follow.  
9. Adhering strictly to the factory pattern may limit opportunities for optimization or alternative design approaches.

**Code**

Tutorialspoint

**Difference with similar pattern**

**Abstract Factory Pattern**: Like the Factory pattern, it centralizes object creation, but it deals with families of related or dependent objects.  
May be prototype but [[ later ]]

**Diagram**

Tutorialspoint