# Factory Method Design Pattern: Theory Answers

## 1. What is the Factory Method Design Pattern and why is it useful?

The Factory Method Design Pattern is a creational pattern that defines an interface for creating an object but allows subclasses to alter the type of objects that will be created. Instead of instantiating objects directly using a constructor (i.e., `new`), the Factory Method delegates the instantiation to subclasses. This pattern is useful because it promotes loose coupling in the codebase, making it easier to introduce new types without changing existing code.

## 2. How does the Factory Method pattern promote flexibility and scalability in object creation?

The Factory Method pattern promotes flexibility by abstracting the object creation process. This means the client code does not need to know the concrete class being instantiated; it only relies on the interface or abstract class. Scalability is achieved because adding new document types (like a new format) only requires creating a new factory class without modifying the core logic of the application. This makes it easier to extend the system as requirements evolve.

## 3. What are the benefits of using a separate factory class for each document type (e.g., Word, PDF, Excel)?

Using a separate factory class for each document type enhances modularity and separation of concerns. Each factory handles the creation logic for a specific document type, which keeps the code organized and easier to maintain. It also adheres to the Single Responsibility Principle, since each class has one job—creating a specific type of document. Furthermore, testing and debugging become simpler because the creation logic is encapsulated within its own factory.

## 4. How is this pattern better than directly instantiating objects using new in large applications?

In large applications, directly using `new` to create objects tightly couples the client code with the concrete classes. This makes the system rigid and harder to maintain or extend. The Factory Method pattern provides an abstraction that hides the instantiation logic and promotes loose coupling. As a result, it becomes easier to change, replace, or add new classes without affecting the client code. It also enhances maintainability and allows for better adherence to SOLID principles.