### **1. Abstract Factory**

* **Structure:** Provides an interface for creating families of related objects without specifying their concrete classes.
* **Characteristics:** Ensures consistent implementation of multiple related products.
* **Implementation:** Uses interfaces or abstract classes to define factory methods and their concrete implementations.
* **Use Case:** When an application needs to work with multiple families of related objects.
* **Advantages:** Removes dependency on concrete classes.
* **Disadvantages:** Leads to complex class hierarchies.

### **2. Builder**

* **Structure: Builds a complex object step by step.**
* **Characteristics: Separates the construction process from the object representation.**
* **Implementation: Defines a Builder class for step-by-step construction.**
* **Use Case: When different variations of an object are required.**
* **Advantages: Flexibility and better code readability.**
* **Disadvantages: Requires multiple classes.**

### **3. Factory Method**

* **Structure: Defines an interface for creating objects, but allows subclasses to determine the specific classes to instantiate.**
* **Characteristics: Supports polymorphism and encapsulation.**
* **Use Case: When the creation of objects needs to vary based on conditions.**
* **Advantages: Easy to extend code.**
* **Disadvantages: May lead to many subclasses.**

### **4. Prototype**

* **Structure: Creates new objects by copying an existing object (cloning).**
* **Use Case: When object creation is expensive or complex.**
* **Advantages: Fast object cloning.**
* **Disadvantages: Complexity when dealing with deep copies.**

### **5. Singleton**

* **Structure: Ensures only one instance of a class exists.**
* **Use Case: Managing access to shared resources, such as logging or configuration settings.**
* **Advantages: Controlled instance access.**
* **Disadvantages: Can create issues with testing and thread safety.**

### **6. Adapter**

* **Structure: Allows incompatible interfaces to work together.**
* **Use Case: When an existing class needs to be used with a different interface.**
* **Advantages: Enables code reuse.**
* **Disadvantages: Adds an extra layer of code.**

### **7. Bridge**

* **Structure: Separates abstraction from implementation.**
* **Use Case: When abstraction and implementation need to evolve independently.**
* **Advantages: Improves code scalability.**
* **Disadvantages: Increases code complexity.**

### **8. Composite**

* **Structure: Combines objects into a tree structure to treat individual and composite objects uniformly.**
* **Use Case: Working with hierarchical object structures.**
* **Advantages: Simplifies the handling of complex structures.**
* **Disadvantages: Can be difficult to design.**

### **9. Decorator**

* **Structure: Dynamically adds functionality to objects without modifying their structure.**
* **Use Case: Adding functionality to objects at runtime.**
* **Advantages: Flexible and extendable.**
* **Disadvantages: Results in many small classes.**

### **10. Facade**

* **Structure: Provides a simplified interface to a complex subsystem.**
* **Use Case: Hiding the complexity of a system behind a simple API.**
* **Advantages: Simplifies system access.**
* **Disadvantages: May reduce flexibility.**

### **11. Proxy**

* **Structure: Provides a placeholder to control access to another object.**
* **Use Case: To control access, optimize, or protect an object.**
* **Advantages: Manages access to resources efficiently.**
* **Disadvantages: Adds an additional layer of code.**

### **12. Chain of Responsibility**

* **Structure: Passes a request through a chain of objects until one handles it.**
* **Use Case: When multiple objects can handle a request, but the handler is unknown in advance.**
* **Advantages: Flexible and extendable.**
* **Disadvantages: No guarantee of request handling.**

### **13. Command**

* **Structure: Encapsulates a request as an object.**
* **Use Case: Decouples the sender of a request from its receiver.**
* **Advantages: Allows for parameterizing and queuing requests.**

### **14. Observer**

* **Structure: Defines a dependency between objects where one object notifies others of state changes.**
* **Use Case: When the state of one object affects others, and they need to stay updated.**
* **Advantages: Promotes loose coupling.**

### **15. Strategy**

* **Structure: Defines a family of algorithms and allows switching between them at runtime.**
* **Use Case: When the algorithm used by a class needs to change dynamically.**
* **Advantages: Enables flexibility without modifying the client.**

### **16. Visitor**

* **Structure: Allows adding new operations to an object structure without modifying its classes.**
* **Use Case: Performing operations on objects within a complex structure without changing their classes.**
* **Advantages: Clean separation of operations and object structure.**