**Creational Design Patterns:**

Creational Patterns used in object creation. These patterns provide various ways of creating objects which improves the flexibility and reusability of the code.

Types of creational patterns:

* Factory
* Abstract Factory
* Builder
* Prototype
* Singleton

**Factory:**

Factory which is an abstract class or an interface for creating objects but sub classes are responsible for creating instances.

Ex1: In various food ordering systems, factory can be used to create different type of pizzas based on the customer order

Ex2: In report generators and converters, factories can be used to create reports or convert one form to another form of report based on the user request.

**Abstract Factory:**

Abstract factory is like an interface or abstract class for creating families of related objects without specifying their concrete subclasses.

Ex1: In Mobile app themes, abstract factories can be used to create different types of objects and their styles in the selected theme.

Ex2: In multi-language applications, this factory can be used for families of objects specific to each language or region.

**Builder:**

This is used to create complex objects from simple objects using step-by-step approaches.

Ex1: In Mobile phone assembly, builders can be used to assemble phones with specific configurations based on the model and user preferences.

Ex2: In house construction, these patterns can be employed to create house objects with these custom specifications.

**Prototype:**

The Prototype design pattern allows cloning or copying of objects, useful when creating new objects is complex or costly, avoiding overhead by using prototype objects for generating new instances.

Ex1: In caching mechanisms, generally caching allows us to reduce the load on databases and disks. To achieve this we can prototype patterns by cloning the databases.

Ex2: In game development, the prototype design pattern can be useful to create game objects. Instead of creating a new game object every time whenever needed, this pattern enables us to clone the existing objects which saves cost and time.

**Singleton:**

The Singleton design pattern ensures a class has only one instance and provides a global access point to that instance, preventing multiple instantiations and allowing shared access across the application.

Ex1: In caching systems, where data is stored temporarily for quick access, a Singleton pattern can be used to manage the cache instance. This ensures that all parts of the application access the same cache and avoids duplicate caching.

Ex2: In applications that need to deal with dates, times, or schedules, a calendar or time manager can be helpful. By using the Singleton pattern, we make sure there's only one instance of this manager, ensuring efficient and consistent handling of date and time