Il existe plusieurs types de design patterns, chacun résolvant des problèmes spécifiques de conception logicielle. Voici une liste plus exhaustive avec des explications sommaires pour certains des design patterns les plus couramment utilisés :

1. **Singleton Pattern** :

- *Objectif* : Garantir qu'une classe n'a qu'une seule instance et fournir un point d'accès global à cette instance.

- *Implémentation* : Utilise une instance statique privée de la classe, un constructeur privé et une méthode publique pour récupérer l'instance unique.

2. **Factory Method Pattern** :

- *Objectif* : Définit une interface pour la création d'un objet, mais laisse les sous-classes alterner les types d'objets à créer.

- *Implémentation* : Une interface ou une classe abstraite définit la méthode de création, que les sous-classes concrètes implémentent.

3. **Abstract Factory Pattern** :

- *Objectif* : Fournir une interface pour créer des familles d'objets liés ou dépendants sans spécifier leurs classes concrètes.

- *Implémentation* : Une interface ou une classe abstraite définit des méthodes pour créer chaque type d'objet, et des implémentations concrètes fournissent ces objets.

4. **Builder Pattern** :

- *Objectif* : Séparer la construction d'un objet complexe de sa représentation, permettant la création d'objets avec différentes configurations.

- *Implémentation* : Utilise un directeur pour orchestrer le processus de construction à l'aide d'un constructeur (builder) spécifique.

5. **Prototype Pattern** :

- *Objectif* : Créer de nouveaux objets en copiant un objet existant, évitant ainsi la création coûteuse d'objets.

- *Implémentation* : L'objet à copier implémente une méthode pour cloner.

6. **Adapter Pattern** :

- *Objectif* : Permet à des interfaces incompatibles de travailler ensemble.

- *Implémentation* : Un adaptateur convertit l'interface d'une classe en une autre interface que le client attend.

7. **Decorator Pattern** :

- *Objectif* : Attacher de nouvelles responsabilités à un objet de manière dynamique, en fournissant une alternative à la sous-classe pour étendre la fonctionnalité.

- *Implémentation* : Utilise la composition au lieu de l'héritage, où les objets sont décorés avec des objets additionnels qui ajoutent des fonctionnalités.

8. **Observer Pattern** :

- *Objectif* : Définir une dépendance "un-à-plusieurs" entre les objets, afin que lorsque l'état d'un objet change, tous ses dépendants soient notifiés et mis à jour automatiquement.

- *Implémentation* : Sujet (l'objet observé) et Observateur (les objets qui observent) avec des méthodes de notification.

9. **Strategy Pattern** :

- *Objectif* : Définir une famille d'algorithmes, les encapsuler et les rendre interchangeables. Permet à un client de choisir l'algorithme approprié à utiliser.

- *Implémentation* : Crée une famille d'algorithmes, les encapsule dans des classes distinctes et les fait varier indépendamment du client qui les utilise.

10. **Command Pattern** :

- *Objectif* : Encapsuler une requête sous forme d'objet, permettant de paramétrer des clients avec différentes requêtes, de mettre en file d'attente les requêtes, et de supporter des opérations annulables.

- *Implémentation* : Commande abstraite avec des classes concrètes pour chaque type de commande.

Il existe bien d'autres design patterns, chacun adapté à des situations spécifiques. Le choix du design pattern dépend des exigences particulières du projet.