**Factory Method**

**CreateNewCar** este o clasă care implementează ICarFactory și este responsabilă pentru crearea de instanțe de mașini în funcție de tipul specificat.

**ICarFactory** este o interfață care definește o metodă pentru crearea de instanțe de mașini. În acest exemplu, CreateNewCar implementează această interfață.

**ICar** este o interfață care definește proprietățile pe care toate mașinile le au în comun. Toate clasele de mașini concrete (ElectricCar, HybridCar și CombustionCar) implementează această interfață și definesc detaliile specifice ale fiecărui tip de mașină. **ElectricCar, HybridCar și CombustionCar** sunt clasele concrete care reprezintă tipurile specifice de mașini. Acestea implementează interfața ICar și definesc detaliile specifice ale fiecărui tip de mașină (cum ar fi capacitățile bateriei pentru mașinile electrice sau puterea motorului pentru mașinile hibride și cu combustie internă).

**Abstract factory**

**Explicații**: IVehicleFactory: Interfața pentru fabrica de vehicule. IMotorcycle și ICar: Interfețele pentru motociclete și mașini. VehicleFactory: Implementarea concrete a fabricii de vehicule. ElectricMotorcycle, CombustionMotorcycle și ElectricCar: Implementările concrete ale motocicletelor și mașinilor.

**Relații**: IVehicleFactory are două metode de creare, CreateCar și CreateMotorcycle. VehicleFactory implementează IVehicleFactory și furnizează implementări concrete pentru crearea de mașini și motociclete. ElectricMotorcycle, CombustionMotorcycle și ElectricCar sunt implementări concrete ale interfețelor IMotorcycle și ICar, fiecare având metodele Start și Stop.

**Singleton**

**SingletonCar**: Aceasta este o clasă care acționează ca un controler în cadrul aplicației, furnizând o cale de acces către funcționalitatea sistemului legată de mașini. Este o clasă derivată din ControllerBase, sugerând că este utilizată pentru a manipula cererile HTTP în cadrul unei aplicații web ASP.NET Core.

**CarSingleton**: Aceasta este o clasă singleton care gestionează crearea de instanțe de mașini. Singleton-ul este implementat folosind un obiect Lazy<T>, asigurând că instanța este creată doar la prima accesare și este partajată între toate apelurile ulterioare.

**Builder**

**Constructorul** clasei **CarBuilder** inițializează un obiect BuilderCar, care este obiectul pe care îl vom construi folosind builder-ul.

**Metodele de adăugare a proprietăților:** Fiecare metodă publică din CarBuilder permite adăugarea unei proprietăți specifice a mașinii (Brand, Model, Year, Price, ElectricPower, BatteryCap). Aceste metode returnează însuși CarBuilder, permițând apelarea metodelor în lanț (metodele pot fi apelate una după alta în același lanț de apel).

**Metoda Build**: Metoda Build este apelată pentru a finaliza construcția obiectului și a returna instanța finală a BuilderCar. Această metodă este responsabilă pentru a întoarce obiectul construit până la acel moment

**Descriere**

**Factory Method:**

**Descrierea**: Factory Method este un pattern de proiectare creational care permite definirea unei interfețe pentru crearea unui obiect, dar lasă subclasele să decidă ce clasă să instanțieze. Acest lucru permite unei clase să își tergiverseze instanțierea către subclase.

**Plusuri și minusuri**:

* **Plusuri**: Flexibilitate în instanțierea obiectelor, permite extinderea și refolosirea codului prin subclasare, decuplarea codului client de codul de creare a obiectelor.
* **Minusuri**: Poate duce la creșterea complexității, necesită o ierarhie de clase pentru fiecare tip de obiect.

**Unde se folosește:** Factory Method este folosit în situații în care o clasă nu poate anticipa clasa de obiecte pe care trebuie să le creeze sau când o clasă dorește să lase subclasele să decidă care obiect să fie creat.

**Diferența**: Diferența principală între Factory Method și celelalte pattern-uri de creare este că Factory Method se concentrează pe instanțierea unei singure clase, în timp ce celelalte pot implica crearea mai multor obiecte interconectate sau complexe.

**Abstract Factory:**

**Descrierea**: Abstract Factory este un pattern de proiectare creational care furnizează o interfață pentru crearea familii de obiecte conexe sau dependente fără a specifica clasele concrete. Acesta definește o interfață pentru crearea unor familii de obiecte, dar lasă subclasele să decidă ce obiecte să creeze.

**Plusuri și minusuri**:

* **Plusuri**: Abstracționează procesul de creare a obiectelor, facilitează schimbările în tipurile de obiecte create, asigură conformitatea între obiecte create în cadrul aceleiași familii.
* **Minusuri**: Poate duce la creșterea complexității, necesită o ierarhie de clase pentru fiecare familie de obiecte.

**Unde se folosește**: Abstract Factory este util atunci când o aplicație trebuie să fie independentă de modalitatea de creare, compoziția și reprezentarea obiectelor.

**Diferența**: Diferența principală între Abstract Factory și Factory Method este că Abstract Factory creează o familie de obiecte, în timp ce Factory Method creează un singur obiect.

**Singleton**:

**Descrierea**: Singleton este un pattern de proiectare creational care garantează că o clasă are doar o singură instanță și oferă un mod de a accesa această instanță din orice altă parte a codului.

**Plusuri și minusuri**:

* **Plusuri**: Asigură că o clasă are o singură instanță, ușurează accesul la această instanță, permite controlul asupra accesului la resursele comune.
* **Minusuri**: Poate duce la suprautilizare și dificultăți în testare, poate afecta performanța în aplicații cu cerințe de performanță ridicate.

**Unde se folosește**: Singleton este utilizat în situații în care trebuie să existe o singură instanță a unei clase în întreaga aplicație, cum ar fi manageri de fișiere, conexiuni la baze de date sau loggere.

**Diferența**: Singleton se diferențiază de celelalte pattern-uri deoarece se concentrează pe instanțierea și accesarea unei singure instanțe a unei clase.

**Builder:**

**Descrierea**: Builder este un pattern de proiectare creational care permite construirea unui obiect complex pas cu pas. Acesta separă construcția unui obiect de reprezentarea sa și permite crearea unor obiecte complexe folosind aceeași logică de construcție.

**Plusuri și minusuri**:

* **Plusuri**: Ușurează construcția de obiecte complexe, permite construcția unui obiect pas cu pas, permite variații ale reprezentării obiectului.
* **Minusuri**: Poate duce la creșterea complexității, necesită implementarea unui builder separat pentru fiecare tip de obiect complex.

**Unde se folosește**: Builder este util în situații în care se dorește construcția de obiecte complexe, cum ar fi obiecte care necesită mulți parametri sau obiecte care sunt construite în etape.

**Diferența**: Diferența principală între Builder și celelalte pattern-uri este că Builder se concentrează pe construcția obiectelor complexe, în timp ce celelalte se concentrează pe instanțierea și accesarea obiectelor.

**Prototype:**

**Descrierea**: Prototype este un pattern de proiectare creational care permite crearea de noi obiecte bazate pe o instanță existentă (prototip). În loc să creezi un obiect de la zero, acest pattern implică clonarea unui obiect existent și modificarea ulterioară a copiei clonate, dacă este necesar.

**Plusuri și minusuri:**

* **Plusuri**: Permite crearea de obiecte complexe cu costuri reduse, reduce dependența de clasele concrete, facilitează adăugarea și eliminarea de tipuri de obiecte dintr-o aplicație.
* **Minusuri**: Poate duce la creșterea complexității în implementare, necesită atenție sporită pentru gestionarea referințelor și a stării obiectelor clonate.

**Unde se folosește**: Prototype este util în situații în care procesul de creare a unui obiect este costisitor sau complex și există deja un obiect similar din care să fie clonat. De asemenea, este util atunci când se dorește crearea de obiecte dinamic la runtime. **Diferența**: Diferența principală între Prototype și celelalte pattern-uri de creare este că Prototype se concentrează pe clonarea obiectelor existente pentru a crea noi obiecte, în timp ce celelalte se concentrează pe instanțierea și configurarea obiectelor.