MODELE DE PROIECTARE SOFTWARE
(SOFTWARE DESIGN PATTERNS)

Clasificare	Model	Definiție	Utilitate	Implementare
1. Creaționale	1.1. Singleton	Model care permite crearea unei singure instanțe a unei clase şi oferă un punct global de acces la această instanță.	Atunci când dorim să avem o singură instanță a unei clase care să fie accesibilă în întregul sistem.	Implică utilizarea unei metode statice pentru a returna instanța şi restricționarea constructorului pentru a fi privat sau protejat.
	1.2. Factory Method	Model ce oferă o interfață pentru crearea de obiecte, dar lasă subclasele să decidă clasa concretă a obiectelor pe care le creează.	Atunci când dorim să delegăm responsabilitatea de creare a obiectelor către clasele derivate, astfel încât să putem extinde sau modifica tipurile de obiecte create fără a modifica codul existent.	Implică definirea unei metode de *fabrică* într-o clasă de bază, iar subclasele vor implementa această metodă pentru a crea obiecte specifice.
	1.3. Abstract Factory	Model care furnizează o interfață pentru crearea familiei de obiecte conexe sau dependente fără a specifica clasele concrete ale acestora.	Atunci când dorim să creăm o familie de obiecte care funcționează împreună şi trebuie să fie compatibile între ele.	Implică definirea unei interfețe abstracte pentru *fabrica* de obiecte şi crearea claselor concrete care implementează această interfață pentru a crea diverse familii de obiecte.
	Alte modele creaționale: Builder, Prototype sau Object Pool.			

Clasificare	Model	Definiție	Utilitate	Implementare
2. Structurale	2.1. Adapter	Model care permite obiectelor cu interfețe incompatibile să lucreze împreună prin intermediul unui obiect adaptor.	Atunci când avem două clase sau obiecte cu interfețe diferite şi dorim să le facem să coopereze.	Implică crearea unui obiect adaptor care să traducă cerințele unei interfețe în cerințele celeilalte interfețe.
	2.2. Composite	Model ce permite tratarea obiectelor individuale şi a grupurilor de obiecte într-un mod uniform, ca pe o structură ierarhică.	Atunci când dorim să tratăm obiectele individuale şi colecțiile de obiecte într-un mod uniform, fără a face distincție între ele.	Implică crearea unei interfețe comune pentru obiectele individuale şi grupurile de obiecte, astfel încât acestea să poată fi tratate într-un mod uniform.
	2.3. Proxy	Design pattern care oferă un substitut sau un înlocuitor pentru un obiect real, controlând accesul la acesta şi oferind funcționalități suplimentare.	Atunci când dorim să controlăm accesul la un obiect sau să adăugăm funcționalități suplimentare înainte, în timpul sau după accesarea obiectului real.	Implică crearea unui obiect proxy care să implementeze aceeaşi interfață ca obiectul real şi să îl înlocuiască în interacțiunea cu utilizatorul.
	Alte modele structurale: Decorator, Facade sau Flyweight.			

Clasificare	Model	Definiție	Utilitate	Implementare
3. Comportamentale	3.1. Observer	Design pattern care defineşte o relație de tip unu-la-mulți (*one-to-many*) între obiecte, astfel încât modificările făcute într-un obiect să fie propagate şi să fie notificate automat tuturor obiectelor dependente.	Atunci când dorim să implementăm un sistem de notificare şi actualizare automată între obiecte, astfel încât modificările făcute într-un obiect să influențeze obiectele dependente.	Implică definirea unui subiect observabil şi a unor observatori care se abonează şi primesc actualizări de la subiect.
	3.2. Strategy	Model care permite definirea şi schimbarea comportamentului unui obiect în timpul execuției, fără a modifica clasa obiectului.	Atunci când dorim să avem multiple strategii sau algoritmi pentru a rezolva aceeaşi problemă şi să putem schimba strategia utilizată fără a modifica codul existent.	Implică definirea unei interfețe comune pentru toate strategiile şi implementarea acestor strategii în clase separate.
	3.3. Template Method	Model ce defineşte scheletul unui algoritm într-o clasă de bază, lăsând clasele derivate să implementeze anumite părți specifice ale algoritmului.	Atunci când avem un algoritm comun, dar anumite părți ale acestuia pot varia între implementări diferite.	Implică definirea unei clase de bază care conține metodele comune şi metode abstracte pe care clasele derivate trebuie să le implementeze.
	Alte modele comportamentale: Iterator, Command sau State.			