COMPOSITION VS. INHERITANCE in OOP

Modelul de programare OOP oferă mai multe beneficii: • ușurința de întreținere - OOP promovează descompunerea sistemelor complexe în obiecte mai mici și mai ușor de gestionat

• extensibilitatea - facilitează adăugarea de noi funcționalități prin intermediul moștenirii și a polimorfismului

• reutilizarea codului - încurajează reutilizarea codului prin intermediul conceptelor de moștenire și compoziție

MOȘTENIREA

AVANTAJE:

- Extensibilitate: permite crearea de clase de bază care conțin funcții comune ce trebuie moștenite de către clasele derivate
- Ajută la maparea obiectelor din lumea reală și relațiile acestora în tipuri abstracte
- Asigură că subclasele urmează o interfață standard.

DEZAVANTAJE:

- crește cuplarea între un tip de bază și tipurile sale derivate. Dacă modifici baza, toate subclasele vor fi afectate.

- întrerupe încapsularea, deoarece metodele și atributele clasei de bază sunt expuse claselor derivate

COMPOZIȚIA

Compoziția în OOP se referă la mecanismul de creare a claselor existente prin includerea instanțelor de obiecte. Aceasta stabilește o relație "has a" între clase, în care o clasă conține un obiect al altei clase.

Compoziția combină clasele existente pentru a crea clase mai complexe, promovând reutilizarea codului. Deoarece obiectele pot fi compuse dinamic în timpul rulării și pot fi înlocuite sau modificate fără a afecta întregul sistem, compoziția permite o mai mare flexibilitate și modularitate.

Când creezi o clasă de bază de la care una sau mai multe clase vor moșteni proprietăți și comportamente, furnizezi o interfață comună pentru subclase. Aceste beneficii vin la costul unor efecte negative:

- Toate subclasele de bază vor fi obligate să respecte implementările interfeței
- Modificarea implementării clasei de bază poate deveni dificilă în timp.
- Cuplajul strâns între clasa de bază și toate subclasele sale poate îngreuna dezvoltarea

Există modalități de rezolvare a acestor probleme cum ar fi utilizarea principiilor SOLID - OOP.
Totuși, o alternativă mai bună este înlocuirea moștenirii cu compoziție, cu excepția cazului în care există un motiv specific pentru utilizarea moștenirii.

Preferăm compoziția în locul moștenirii • pentru a evita cuplarea strânsă a codului. Amintește-ți regulile "is a" și "has a" când proiectezi tipurile.

• pentru reutilizarea codului și că utilizează interfețe pentru polimorfism fără relația "is a"

Decizia depinde de cerințe. Combinația între compoziție și moștenire poate fi o alegere bună în dezvoltarea sistemelor.