Kompozycja w Javie

Maciej Borowy 2P

Definicja:

Kompozycja w Javie to relacja typu „ma-a”, która umożliwia ponowne wykorzystanie kodu w programach. Oznacza to, że jeden obiekt jest częścią innego obiektu, co pozwala na elastyczność w implementacji klas, umożliwiając zmianę dołączonego obiektu w czasie wykonywania. Kompozycja jest bardziej specyficznym rodzajem agregacji, co oznacza, że jeden obiekt jest logicznie większą strukturą, która zawiera drugi obiekt.

Zalety:

•**Elastyczność** – Kompozycja daje większą elastyczność w tworzeniu obiektów. Można łatwo zmieniać klasy, które są częścią innych klas, bez konieczności modyfikowania dziedziczenia.
•**Zwiększona spójność** – Kompozycja pozwala na tworzenie spójnych obiektów, ponieważ można tworzyć złożone obiekty z mniejszych, dobrze zdefiniowanych komponentów.
•**Brak problemów z wielokrotnym dziedziczeniem** – W przeciwieństwie do dziedziczenia, kompozycja nie prowadzi do problemów związanych z dziedziczeniem po wielu klasach.
•**Luźniejsze powiązanie** – Kompozycja jest mniej powiązana z konkretną implementacją.

Wady:

•**Zwiększenie złożoności** – Tworzenie wielu obiektów składających się z innych obiektów może wprowadzać większą złożoność w kodzie, ponieważ każdy obiekt wymaga odpowiedniej inicjalizacji.
•**Potrzebna jest odpowiednia organizacja** – W przypadku złożonych struktur kompozycji może być konieczne zachowanie dużej dyscypliny przy zarządzaniu zależnościami między obiektami.
•**Potrzebna jest kontrola cyklu życia obiektów** – W przypadku bardziej złożonych struktur, zarządzanie cyklem życia obiektów (np. obiektów współdzielonych) może być trudne.

Przykłady kompozycji(1):

```
class Silnik {
    public void uruchom() {
        System.out.println("Silnik uruchomiony.");
    }
}

class Samochod {
    private Silnik silnik = new Silnik(); // Kompozycja

    public void uruchom() {
        silnik.uruchom();
        System.out.println("Samochód uruchomiony.");
    }
}

public class Main {
    public static void main(String[] args) {
        Samochod samochod = new Samochod();
        samochod.uruchom();
    }
}
```

Przykłady kompozycji(2):

```java
class Procesor {
    public void uruchom() {
        System.out.println("Procesor uruchomiony.");
    }
}

class PamiecRAM {
    public void zaladujDane() {
        System.out.println("Dane załadowane do pamięci.");
    }
}

class Komputer {
    private Procesor procesor = new Procesor();
    private PamiecRAM pamiecRAM = new PamiecRAM();

    public void uruchomKomputer() {
        procesor.uruchom();
        pamiecRAM.zaladujDane();
        System.out.println("Komputer uruchomiony.");
    }
}

public class Main {
    public static void main(String[] args) {
        new Komputer().uruchomKomputer();
    }
}
```

Dlaczego jest trend zamieniania dziedziczenia na kompozycję?:

Trend zamiany dziedziczenia na kompozycję wynika z potrzeby większej ***elastyczności i łatwiejszego utrzymania kodu***. Kompozycja pozwala na ***luźniejsze powiązania*** między klasami, co ułatwia zmiany w systemie bez wpływu na resztę aplikacji. Dziedziczenie, zwłaszcza w głębokich hierarchiach, może prowadzić do problemów z niejednoznacznością (np. wielokrotne dziedziczenie) i utrudniać modyfikacje.

Kompozycja wspiera zasadę pojedynczej odpowiedzialności, umożliwiając tworzenie modułowych komponentów, co poprawia przejrzystość kodu. Dodatkowo, łatwiej jest testować aplikację z użyciem kompozycji, ponieważ komponenty można łatwo podmieniać. Zmiana komponentu w systemie kompozycyjnym jest także mniej inwazyjna, w przeciwieństwie do dziedziczenia, gdzie zmiany w klasach bazowych mogą wymagać modyfikacji wielu klas pochodnych.

W skrócie, kompozycja daje większą elastyczność, modularność i ułatwia rozwój systemów, podczas gdy dziedziczenie często prowadzi do sztywnych zależności i trudniejszych do utrzymania hierarchii klas.

Dziękuję za uwagę