|  |  |
| --- | --- |
| OOPs Design Principles | |
| Favoring Composition Over Inheritance In Java | |
| Favoring Composition over Inheritance is a principle in object-oriented programming (OOP). Classes should achieve polymorphic behavior and code reuse by their composition rather than inheritance from a base or parent class. To get the higher design flexibility, the design principle says that composition should be favored over inheritance.  Inheritance should only be used when subclass ‘is a’ superclass. Don’t use inheritance to get code reuse. If there is no ‘is a’ relationship, then use composition for code reuse. | |
| REASONS | 1. The fact that Java does not support multiple inheritances is one reason for favoring composition over inheritance in Java. Since you can only extend one class in Java, but if you need multiple features, such as reading and writing character data into a file, you need Reader and Writer functionality. It makes your job simple to have them as private members, and this is called Composition. 2. Composition offers better test-ability of a class than Inheritance. If one class consists of another class, you can easily construct a Mock Object representing a composed class for the sake of testing. This privilege is not given by inheritance. 3. Although both Composition and Inheritance allow you to reuse code, one of Inheritance’s disadvantages is that it breaks encapsulation. If the subclass depends on the action of the superclass for its function, it suddenly becomes fragile. When super-class behavior changes, sub-class functionality can be broken without any modification on its part. 4. In the timeless classic Design Patterns, several object-oriented design patterns listed by Gang of Four: Elements of Reusable Object-Oriented Software, favor Composition over Inheritance. Strategy design pattern, where composition and delegation are used to modify the behavior of Context without touching context code, is a classic example of this. Instead of getting it by inheritance, because Context uses composition to carry strategy, it is simple to have a new implementation of strategy at run-time. 5. Another reason why composition is preferred over inheritance is flexibility. If you use Composition, you are flexible enough to replace the better and updated version of the Composed class implementation. One example is the use of the comparator class, which provides features for comparison. |