Композиция в Java

Композиция – это создание в нашем классе объектов других классов (*для возможности вызывать методы этих классов в нашем классе*) до написания кода для метода/ов нашего класса:

Class Aaa

{

**Bbb b = new Bbb();**

**Ccc c = new Ccc();**

- - - - - - - - - - - - - - -

metod()

{

- - - - - - - - - - - - - - -

b.metodClassaBbb();

- - - - - - - - - - - - - - -

c.metodClassaCcc();

- - - - - - - - - - - - - - -

}

}

Чем отличаются наследование и композиция в Java

Несмотря на то, что и *композиция*, и *наследование* позволяют использовать код повторно, они делают это по-разному. Основное отличие между ними состоит в том, что композиция позволяет переиспользовать код без его расширения. Наследование при этом требует расширения существующего класса. Другое важное отличие: при композиции мы можем повторно использовать код даже из final-класса, тогда как унаследоваться от него невозможно. Кроме того при композиции возможно использование кода из нескольких различных классов. С наследованием такой трюк не сработает: в Java не поддерживается множественное наследование. Зато вы можете сделать это в C++. Кстати, всегда стоит предпочитать композицию наследованию в Java. И не только потому, что так советует Джошуа Блох в своей книге [Effective Java](https://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20" \t "_blank) (которая является отличным источником полезных советов для Java-программиста). Аргументы за использование композиции вы можете прочитать [здесь](https://javarevisited.blogspot.sg/2013/06/why-favor-composition-over-inheritance-java-oops-design.html).

Наследование и композиция

Давайте посмотрим более детально на отличия композиции и наследования. Мы рассмотрим каждый пункт достаточно подробно, но без скучных деталей.

Гибкость

Первое отличие касается гибкости кода. При использовании наследования вы должны описать, какой класс вы расширяете. При этом, вы не можете заменить его во время выполнения программы. Используя композицию, вы можете определить используемый тип, который может содержать несколько различных реализаций. Таким образом, композиция предоставляет гораздо больше гибкости, чем наследование.

Ограниченное повторное использование кода при наследовании

Как уже было сказано, унаследоваться в Java можно только от одного класса, а это значит, что повторно можно использовать один, и только один класс. Если вы хотите получить функциональность нескольких классов, то вам следует использовать композицию. К примеру: ваш код должен использовать аутентификацию, следовательно вам надо унаследовать класс Authenificator, для использования авторизации — Autorizer и т. д. Но, поскольку [множественное наследование в Java не поддерживается](https://javarevisited.blogspot.sg/2011/07/why-multiple-inheritances-are-not.html), вы этого сделать не можете. Единственный выход — композиция.

Юнит-тесты

Важный аргумент при выборе способа повторного использования кода состоит в том, что классы, расширенные при помощи композиции, легче тестировать, поскольку вы всегда можете предоставить заглушку для используемого класса. При наследовании вам потребуется родительский класс для тестирования, и заменить его заглушкой не получится.

Final-классы

Невозможность расширить final-классы — еще одно ограничение наследования. В Java [нет возможности унаследоваться от final-классов](https://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html), поэтому опять единственный выход — использовать композицию для повторного использования кода.

Инкапсуляция

Последнее отличие композиции и наследования в Java заключается в их отношении к инкапсуляции. Несмотря на то, что и та, и другая техники позволяют повторно использовать код, наследование нарушает принцип инкапсуляции, поскольку подкласс зависит от поведения родительского класса. Если класс-родитель изменит свое поведение, это повлияет и на его потомков. Если классы плохо документированы и класс-потомок неправильно использует родительский, то при любом изменении родителя функциональность потомка окажется сломанной. Для того, чтобы увидеть это на примере, вы можете прочитать главы [16 и 17 Effective Java](https://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20).

Вот и все, что можно сказать об отличиях композиции и наследования в Java и в объектно-ориентированном программировании вообще. Как видите, они по-разному служат одной цели: повторное использование проверенных и протестированных участков кода. Композиция при этом позволяет защитить повторно используемый класс от клиентов, тогда как наследование этого не гарантирует. Тем не менее, иногда наследование необходимо. В частности, когда вы создаете классы из одного семейства.

Также использовать наследование нужно, когда я хочу, чтобы при изменении класса родителя менялось и поведение класса наследника.