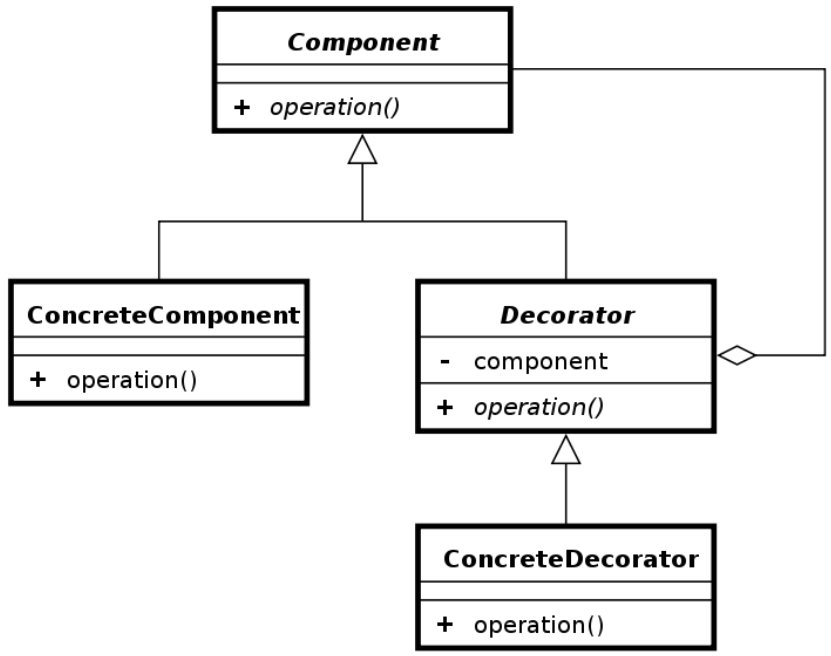
Шаблон Decorator

Шаблонът **Декоратор** добавя функционалност към вече съществуващ клас. Известен е още като "**Wrapper**" и предоставя алтернатива на наследяването. Този шаблон взима вече съществуващ клас и го „обвива" като добавя нова функционалност. Декоратора пренасочва всички заявки към обвития клас и може да извършва допълнителна работа, преди или след самото пренасочване. Декораторите могат да се **nest**-ват или пъхат един в друг рекурсивно. Така всеки нов декоратор обвиващ вече съществуващ декоратор може да добавя допълнителна функционалност. В .**Net** статичният клас **Console** ни предоставя възможност за писане на конзолата. Ако искаме да изведем цветен ред на конзолата трябва да зададем какъв цвят да се използва, да изведем реда и накрая да върнем първоначалния цвят за да не повлияе тази промяна на по-нататъшното изпълнение на програмата. Слагането и връщането на цвят е доста монотомна задача, особенно ако се изпълнява много пъти и на различни места в кода. За това оригиналният клас **Console**, може да се обвие в класа **ColorConsole**, който ще пренасочва всички заявки за писане на ред като добавя възможност за автоматично слагане и връщане на цвят. Моля погледнете в **01. Decorator** за примерна реализация.

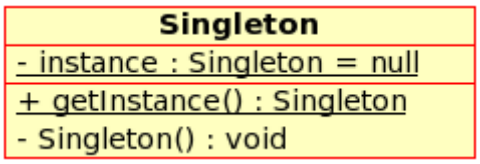


Фигура 1. Клас диаграма на **Decorator** шаблона.

Шаблон Singleton

Този шаблон предоставя една глобална инстанция с единствена точка за достъп за всички обекти на приложението. Не трябва да се бърка с глобална променлива. **Singleton** позволява във всеки един момент да има само една инстанция, която да се досъпва лесно, но също така пречи за създаването на нови инстанции от този обект. В .**Net** класа **Random** позволява за произволно генереиране на число в даден диапазан. Въпреки това операцията за произволност е базираната на конкретни стъпки. При инициализация на обект от тип **Random** се взема така

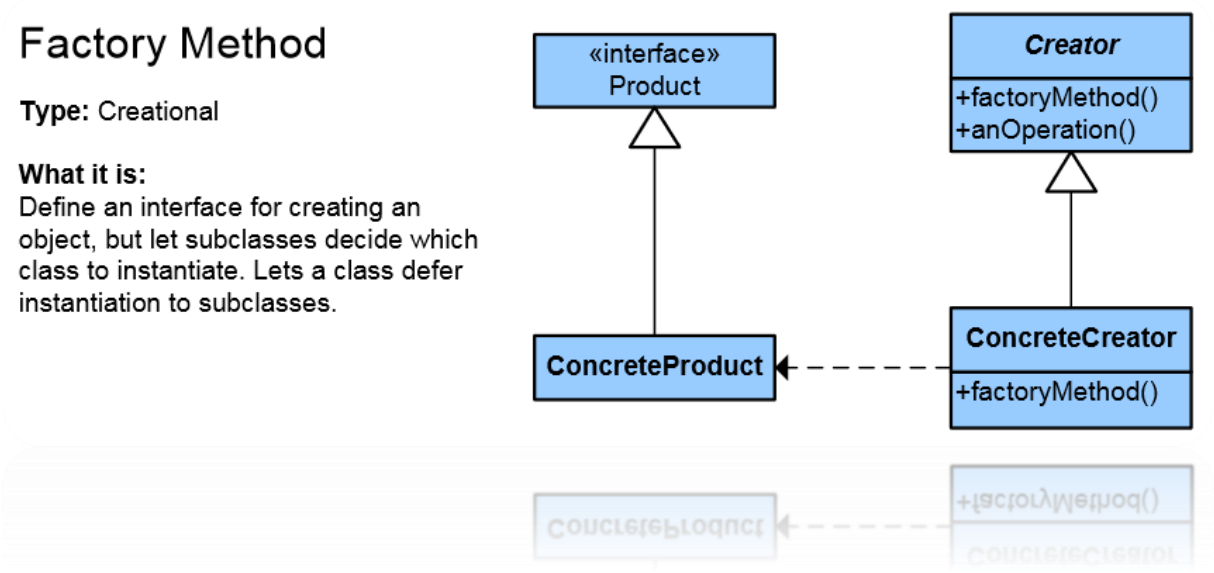
наречения **seed** от системния часовник или това е момента, в който е генериран този обект. На базата на този **seed** се връща произволно число, и след това се подменя защото вече е бил използван. Проблема идва когата много бързо се генерира нови и нови променливи от този тип, чиити **seed** ще е еднакъв докато показанията на системния часовник не се променят. Така върнатите „*произволни*“ стойности ще са еднакви защото са базирани на еднакъв **seed**. **Singleton** шаблона ни гарантира че в цялото приложение ще има точно един обект от даден тип, затова може да го приложим към **Random** създавайки си нов клас **RandomGenerator,** чиито **seed** се взема точно веднъж, когато инициализираме класа за първи път. Моля погледнете в **02. Singleton** за примерна реализация.



Фигура 2. Клас диаграма на **Singleton** шаблона.

Шаблон Factory

Този шаблон помага за създаването на обекти имплементиращи един и същи **интерфейс**. Създаването на обекти става на отделно място (клас или метод), като скрива сложната логика при инициализирането на обекти. Това добринася за по-голяма абстракция и преизползваемост на кода, понеже обектите могат да се подменят стига да имплементират конкретният интерфейс. Например в играта **Tetris** прозиволните фигури могат да се генерират на едно централно място. Моля погледнето в **03. SimpleFactory** за примерна реализация.



Фигура 3. Клас диаграма на **Factory** шаблона.