Как два программиста хлеб пекли

Программирование*, ООП*

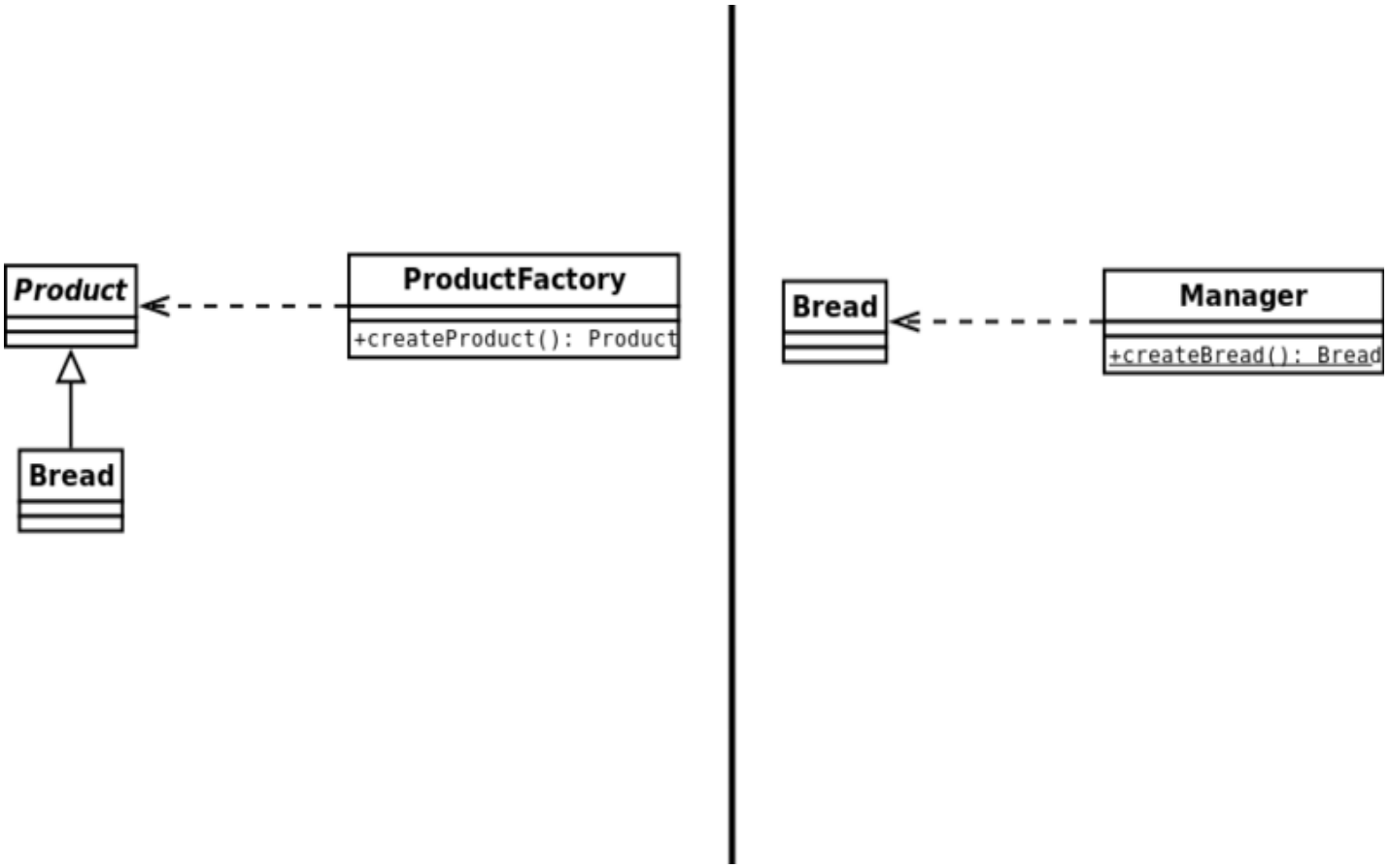
Я работаю программистом уже много лет, на протяжении которых, как это ни странно, я всё время что-то программирую. И вот какую интересную вещь я заметил: в коде, написанном мной месяц назад, всегда хочется что-то чуть-чуть поправить. В код полугодичной давности хочется поменять очень многое, а код, написанный два-три года назад, превращает меня в эмо: хочется заплакать и умереть. В этой статье я опишу два подхода. Благодаря первому архитектура программы получается запутанной, а сопровождение — неоправданно дорогим, а второй — это принцип KISS.

Итак, представим себе, что есть два программиста. Один из них умный, прочёл кучу статей на Хабре, знает каталог GoF наизусть, а Фаулера— в лицо. Другой же делает всё просто. Первого будут звать, например, Борис Н., а второго — Маркус П. Само собой, имена вымышленные, и все совпадения с реальными людьми и программистами случайны.

Итак, к ним обоим приходит проектный менеджер (если в вашей вселенной PM не ходит сам к программистам, назовите его как-то иначе, например BA или lead, сути это не изменит) и говорит:
— Ребята, нам нужно, чтобы делался хлеб.

Именно так, «делался», без уточнения способа производства.

Как же поступят наши программисты?

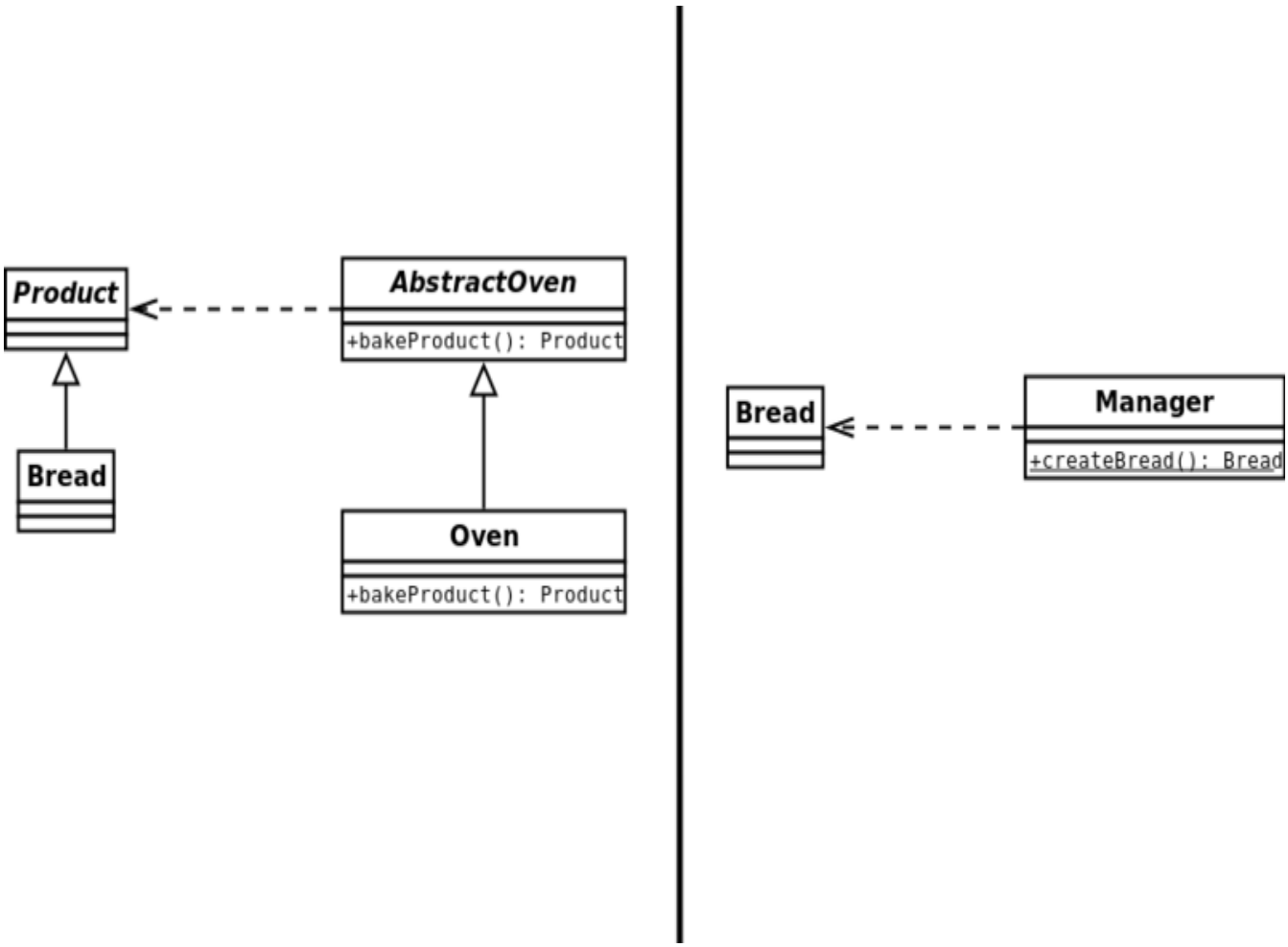


Борис создаёт свою первую абстракцию — класс Product, от него он наследует класс Bread, а инстанциирует экземпляры этого класса фабричный метод класса ProductFactory — createProduct().

Маркус делает примерно то же. Он создаёт класс Bread и класс Manager с фабричным методом createBread().

Пока разница минимальна. Проектный менеджер, чуть глубже разобравшись (это ему так только кажется, да) в потребностях заказчика, приходит во второй раз и говорит:
— Нам нужно, чтобы хлеб не просто делался, а выпекался в печке.

А сразу нельзя было сказать, что хлеб печётся не в вакууме, а в печке? Ну ладно, что же делают программисты?

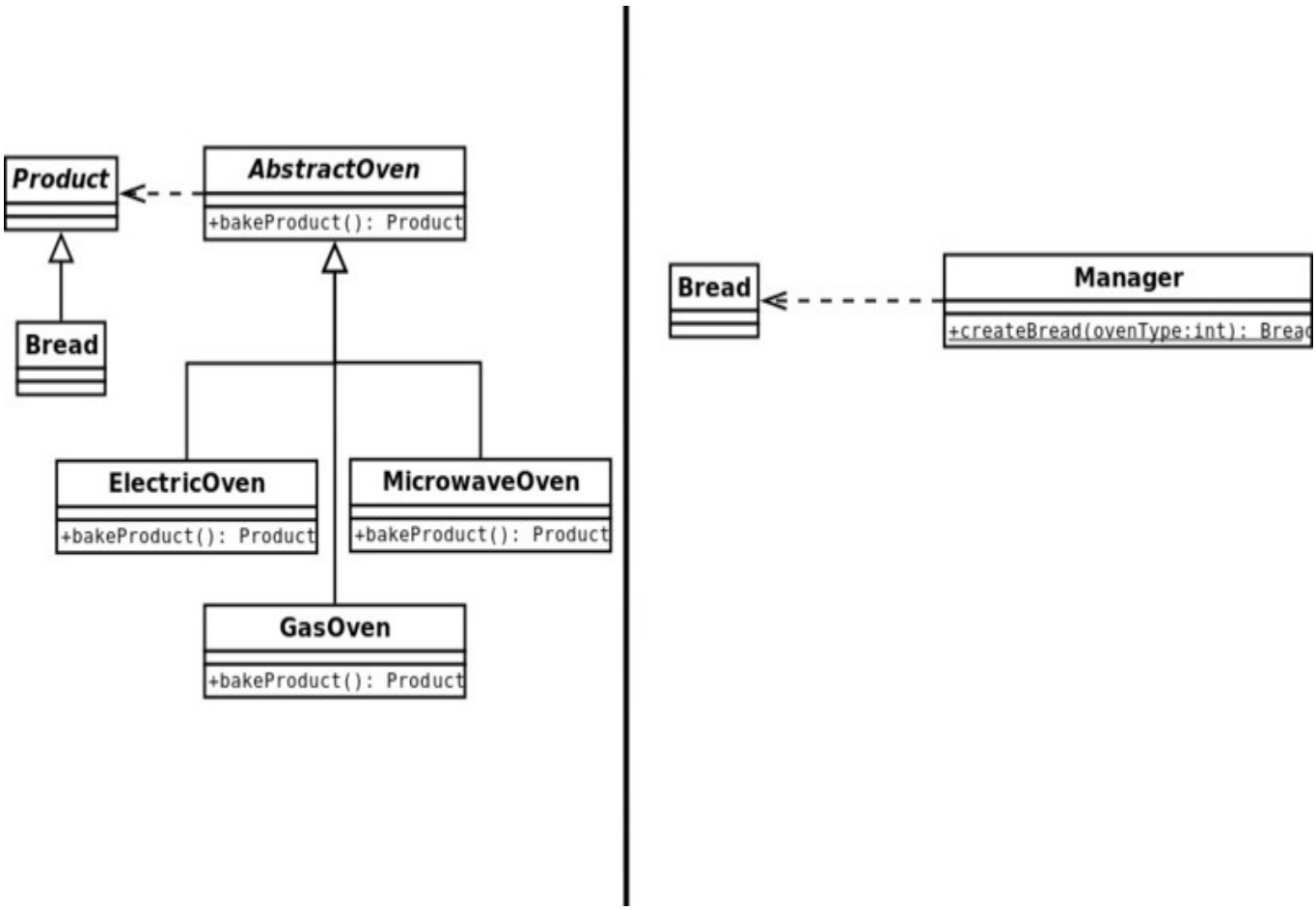


Борис переименовывает класс ProductFactory в Oven, и выделяет абстракцию — AbstractOven. Чтобы было совсем красиво, он метод createProduct() переименовывает в bakeProduct(). Тем самым Борис в первый раз выполнил рефакторинг, применив «выделение абстракции», а так же реализовал шаблон «абстрактная фабрика» точь-в-точь как он описан в литературе. Молодец, Борис.

А вот Маркус ничего не делает. С его точки зрения всё и так хорошо. Ну может быть, стоит слегка поменяет реализацию createBread().

Фаза луны меняется, и менеджер в третий раз приходит к программистам. Он говорит:
— Нам нужно, чтобы печки были разных видов.

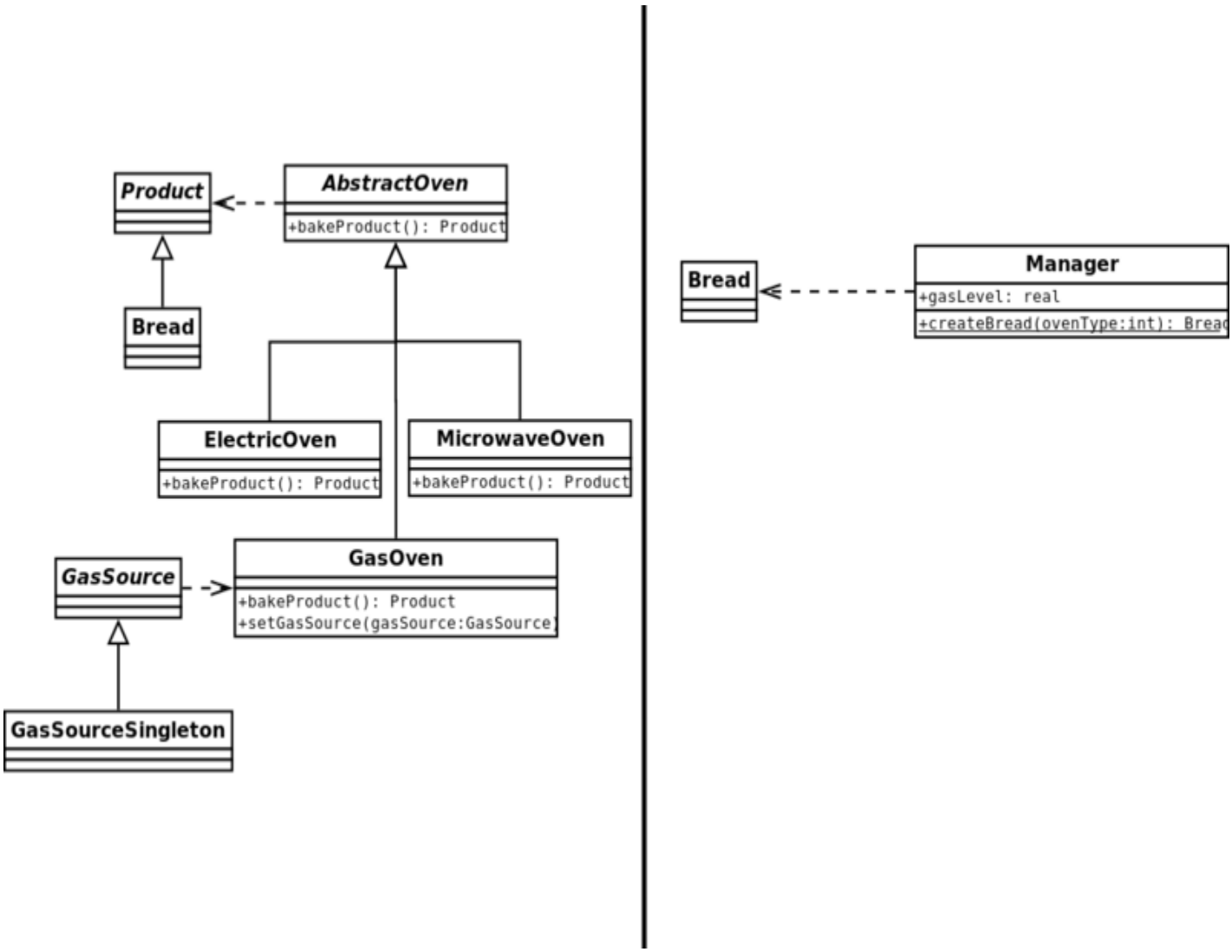
Что ж, справедливо.



Борис, радостно потирая руки, создаёт три наследника AbstractOven — ElectricOven, MicrowaveOven и GasOven. А класс Oven он удаляет за ненужностью.

Маркус тоже вносит изменения в программу. Он добавляет в метод createBread целочисленный параметр ovenType.

В четвёртый раз приходит к программистам менеджер. Он только что прочёл одну из книг серии «Я познаю мир». Интерференция новой информации и PMBoK дала неожиданный результат. Менеджер говорит:
— Нам нужно, чтобы газовая печь не могла печь без газа.

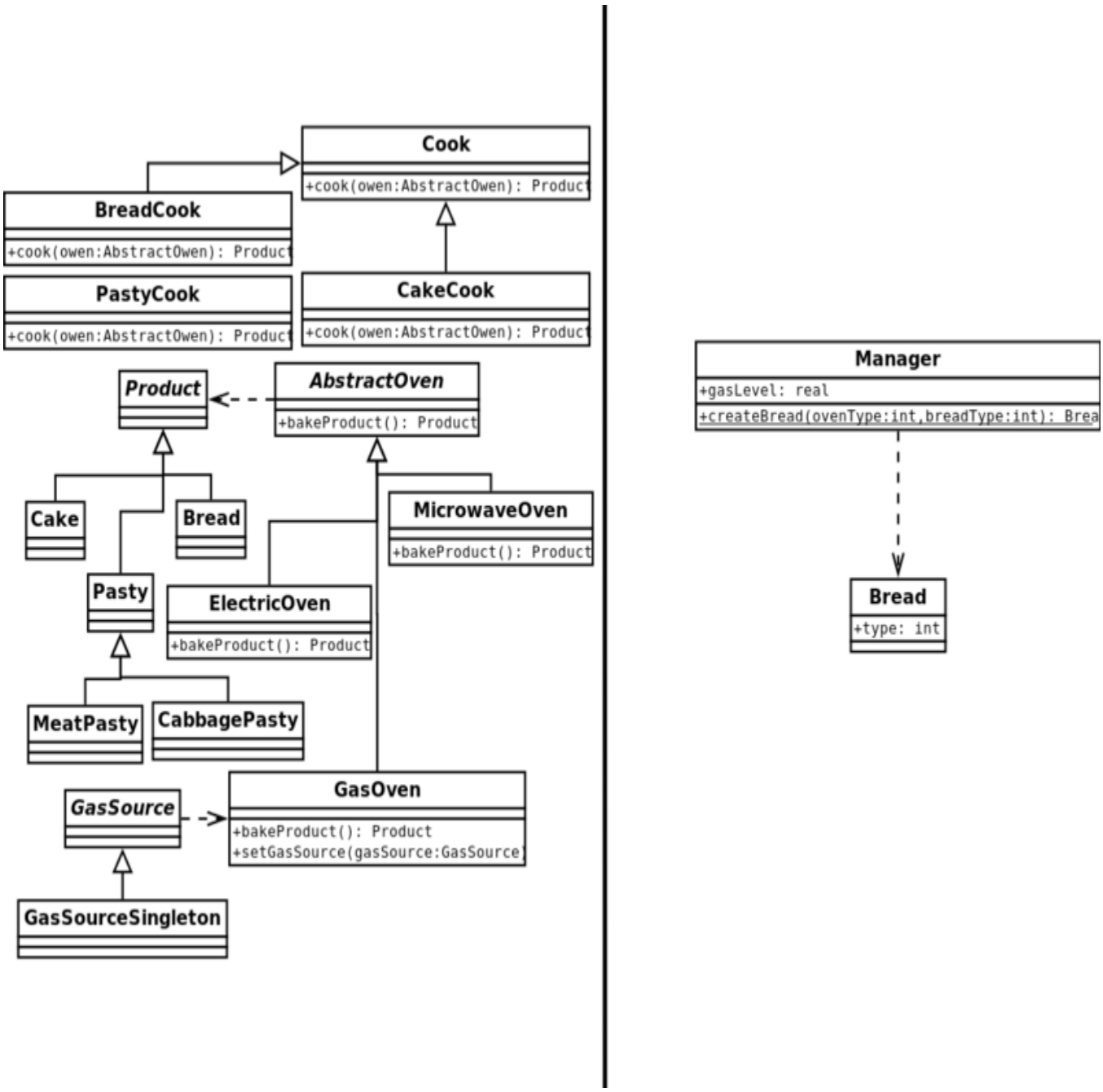


Борис абсолютно безосновательно считает, что источник газа может быть только один. А для таких случаев всегда есть наш любимый шаблон. Он создаёт одиночку GasSourceSingleton, а для уменьшения связности внедряет его через интерфейс GasSource в GasOven. Ура, он применил внедрение зависимости через сеттер!

Скромный от природы Маркус создаёт вещественное приватное поле gasLevel в классе Manager. Естественно, придётся чуть поменять логику метода createBread, но что поделаешь!

Но вот пару дней спустя менеджер приходит в пятый раз, и, сыто облизываясь, произносит:
— Нам нужно, чтобы печки могли выпекать ещё и пирожки (отдельно — с мясом, отдельно — с капустой), и торты.

Программисты тоже хотят есть, поэтому берутся за работу.

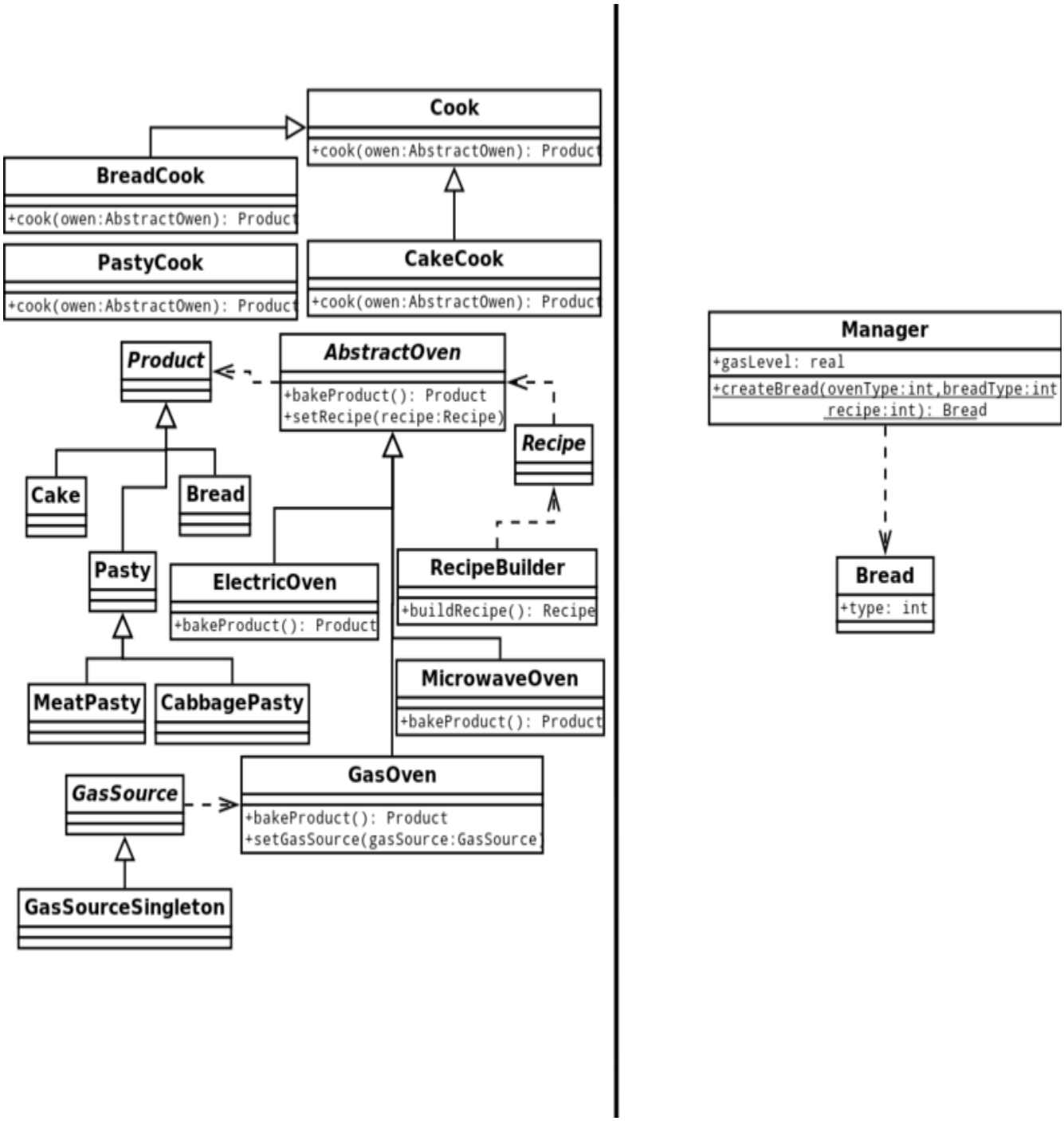


Борис уже начинает что-то такое чувствовать, но остановиться уже не может. Как печка узнает, что именно ей нужно готовить? Очевидно же — ей нужен повар. И Борис, не долго (а может и долго) думая, создаёт класс Cook. У него будет метод для приготовления, принимающий на вход абстрактную печь — cook(owen: AbstractOwen): Product. Ведь это логично — повар берёт печь, и с её помощью готовит. Потом Борис создаёт ещё несколько наследников класса Product — Cake и Pasty, а от Pasty наследует MeatPasty и CabbagePasty. А затем для каждого типа продукта создаёт отдельного повара — BreadCook, PastyCook и CakeCook.

Вроде ещё нормально, но времени на это ушло намного больше, чем у Маркуса, который просто добавил ещё один целочисленный параметр к методу createBread — breadType.

В шестой раз приходит менеджер. Кстати, то, что он сейчас попросит — это не требование заказчика, это его собственная инициатива. Но ведь об этом никто не узнает, так ведь?

— Нам нужно, чтобы хлеб, пирожки и торты выпекались по разным рецептам.

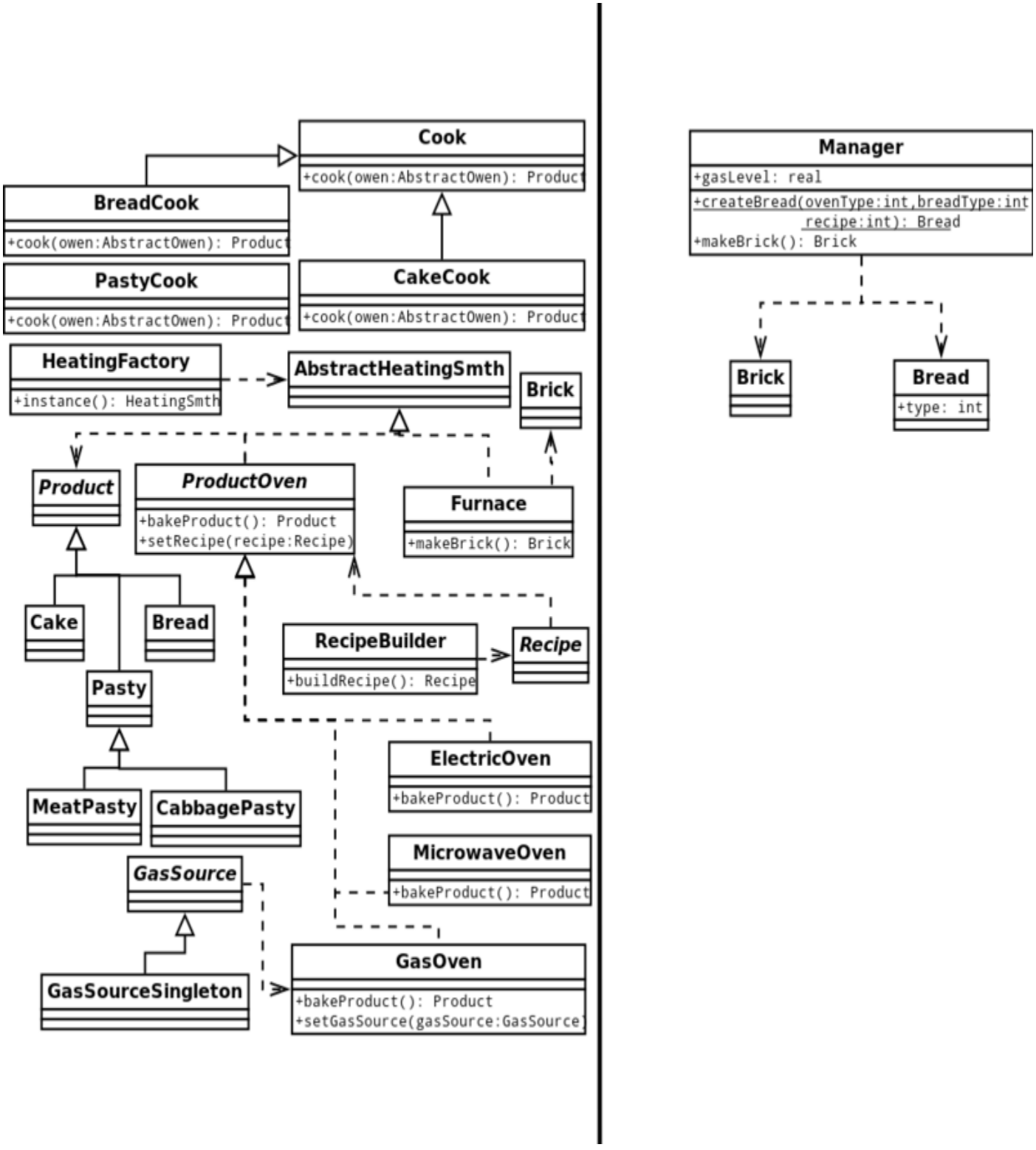


«Хм», — произносит Борис и вспоминает про шаблон «строитель» (вместе со «свободным интерфейсом», конечно же). Он создаёт класс Recipe, а к нему — строитель RecipeBuilder. Рецепт он внедряет (ВНЕЗАПНО!) в печку с помощью сеттера setRecipe(recipe:Recipe).

А Маркус (вы не поверите) добавляет ещё один целочисленный параметр в createBread — recipe.

Самое интересное, как всегда, происходит вдали от компьютеров. А именно: менеджер впервые после начала разработки встречается с заказчиком и наконец-то понимает, зачем тому нужна была печка. Он (менеджер) в седьмой раз приходит к программистам и

говорит:
— Нам нужно, чтобы в печи можно было обжигать кирпичи.



Для Бориса это последняя встреча с менеджером, но всё же он из последних сил вносит изменения в архитектуру. Он выделяет абстрактный класс AbstractHeatingSmth — абстрактное нагревающее нечто. Для него он создаёт фабрику HeatingFactory. От AbstractHeatingSmth он наследует ProductOven и Furance. У последнего есть фабричный метод makeBrick, создающий экземпляр объекта Brick. Но ничего не работает. Читателю

предлагается самостоятельно найти ошибку в архитектуре.

У Маркуса тоже не всё так гладко. Ему приходится создать уже третий (!) по счёту класс. Он называет его Brick, и добавляет в свой Manager метод makeBrick.