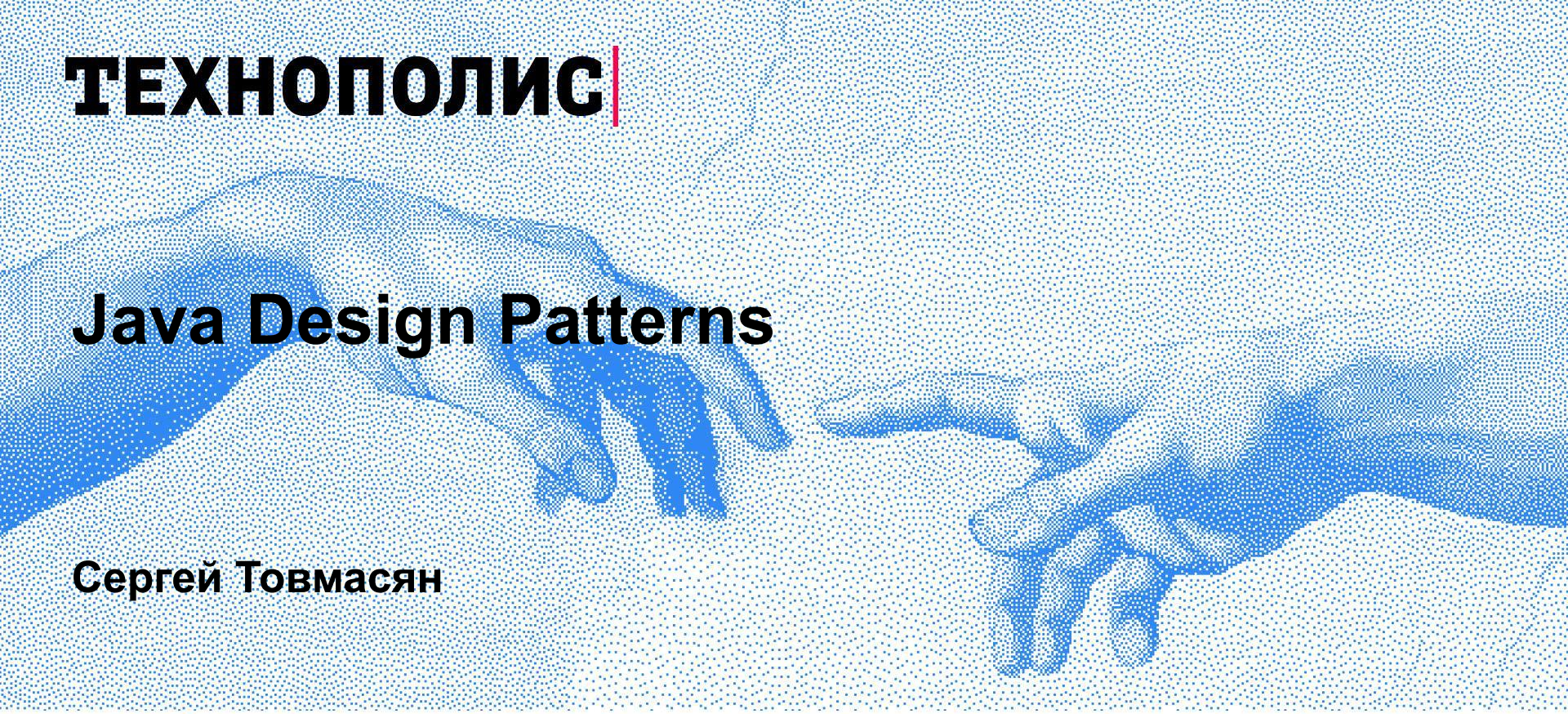
ТЕХНОПОЛИС

Java Design Patterns



Сергей Товмасян

одноклассники

ПОЛИТЕХ

@mail.ru group

Паттерны проектирования

Шаблоны проектирования — это проверенные и готовые к использованию решения часто возникающих в повседневном программировании задач. Это не класс и не библиотека, которую можно подключить к проекту, это нечто большее. Шаблон проектирования, подходящий под задачу, реализуется в каждом конкретном случае

То есть это повторимая архитектурная конструкция.

Lazy initialization – без реализации

Иногда требуется что-то иметь под рукой, на всякий случай, но не всегда хочется прилагать каждый раз усилия, чтобы это каждый раз получать/создавать. Для таких случаев используется паттерн «отложенная инициализация». Допустим вы работаете в бухгалтерии и для каждого сотрудника вы должны подготавливать «отчет о выплатах». Вы можете в начале каждого месяца делать этот отчет на всех сотрудников, но некоторые отчеты могут не понадобиться, и тогда скорее всего вы примените «отложенную инициализацию», то есть вы будете подготавливать этот отчет только тогда, когда он будет запрошен начальством (вышестоящим объектом), однако начальство по сути в каждый момент времени может сказать что у него этот отчет уже есть, однако готов он уже или нет, оно не знает и знать не должно. Как вы уже поняли, данный паттерн служит для оптимизации ресурсов.

Singleton

Представьте, что в городе требуется организовать связь между жителями. С одной стороны мы можем связать всех жителей между собой протянув между ними кабели телефонных линий, но полагаю вы понимаете насколько такая система неверна. Например, как затратно будет добавить еще одного жителя в связи (протянуть по еще одной линии к каждому жителю). Чтобы этого избежать, мы создаем телефонную станцию, которая и будет нашим «одиночкой». Она одна, всегда, и если кому-то потребуется связаться с кем-то, то он может это сделать через данную телефонную станцию, потому что все обращаются только к ней. Соответственно для добавления нового жителя нужно будет изменить только записи на самой телефонной станции. Один раз создав телефонную станцию все могут пользоваться ей и только ей одной, в свою очередь эта станция помнит всё что с ней происходило с момента ее создания и каждый может воспользоваться этой информацией, даже если он только приехал в город. Пример в JDK: java.lang.Runtime

Registry – без реализации

Как следует из названия, данный паттерн предназначен для хранения записей которые в него помещают и соответственно возвращения этих записей (по имени) если они потребуются. В примере с телефонной станцией, она является реестром по отношению к телефонным номерам жителей.

Паттерны «одиночка» и «реестр» постоянно встречаются нам в повседневной жизни. Например бухгалтерия в фирме является «одиночкой», потому как она всегда одна и помнит что с ней происходило с момента ее начала работы. Фирма не создает каждый раз новую бухгалтерию когда ей требуется выдать зарплату. В свою очередь бухгалтерия является и «реестром», потому как в ней есть записи о каждом работнике фирмы.

Multiton

Как понятно из названия паттерна, это по своей сути «реестр» содержащий несколько «одиночек», каждый из которых имеет своё «имя» по которому к нему можно получить доступ.

Object Pool – без реализации

По аналогии с «пулом одиночек» данный паттерн также позволяет хранить уже готовые объекты, однако они не обязаны быть «одиночками».

Factory

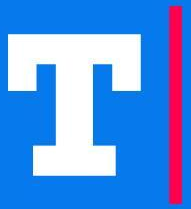
Суть паттерна практически полностью описывается его названием. Когда вам требуется получать какие-то объекты, например пакеты сока, вам совершенно не нужно знать как их делают на фабрике. Вы просто говорите «сделайте мне пакет апельсинового сока», а «фабрика» возвращает вам требуемый пакет. Как? Всё это решает сама фабрика, например «копирует» уже существующий эталон. Основное предназначение «фабрики» в том, чтобы можно было при необходимости изменять процесс «появления» пакета сока, а самому потребителю ничего об этом не нужно было сообщать, чтобы он запрашивал его как и прежде.

Как правило, одна фабрика занимается «производством» только одного рода «продуктов». Не рекомендуется «фабрику соков» создавать с учетом производства автомобильных покрышек. Как и в жизни, паттерн «фабрика» часто создается «одиночкой».

Builder

Данный паттерн очень тесно переплетается с паттерном «фабрики». Основное различие заключается в том, что «строитель» внутри себя, как правило, содержит все сложные операции по созданию объекта (пакета сока). Вы говорите «хочу сока», а строитель запускает уже целую цепочку различных операций (создание пакета, печать на нем изображений, заправка в него сока, учет того сколько пакетов было создано и т.п.). Если вам потребуется другой сок, например ананасовый, вы точно также говорите только то, что вам нужно, а «строитель» уже позаботится обо всем остальном (какие-то процессы повторит, какие-то сделает заново и т.п.). В свою очередь процессы в «строителе» можно легко менять (например изменить рисунок на упаковке), однако потребителю сока этого знать не требуется, он также будет легко получать требуемый ему пакет сока по тому же запросу.

java.lang.StringBuilder#append() (unsynchronized),
java.lang.StringBuffer#append() (synchronized)



Спасибо за внимание!