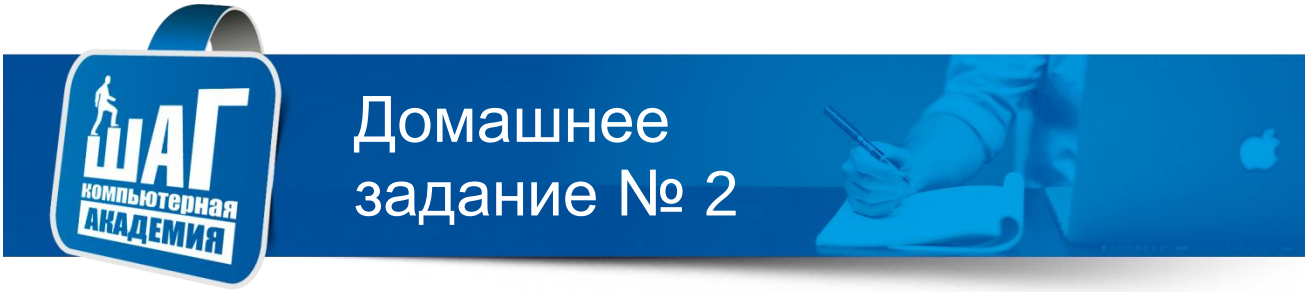

Курс: «Паттерны проектирования»

Тема: Порождающие паттерны.

Fluent Builder

Создаваемая нами программа использует класс, описывающий резюме соискателей вакантной должности. Резюме должно содержать следующую информацию о соискателе: фамилия, имя, год рождения, телефон, e-mail, образование, список компетенций, список ранее занимаемых должностей. Не для всех соискателей необходимо заполнять все данные о соискателе (может отсутствовать опыт предыдущей работы, список компетенций может бать разным).

Для создания объектов в подобных ситуациях традиционно программисты использовали 2 подхода:

1) Паттерн ***«Телескопический конструктор» (Telescoping Constructor)***. Суть этого паттерна состоит в том, что Вы предоставляете несколько конструкторов: конструктор с обязательными параметрами, конструктор с одним дополнительным параметром, конструктор с двумя дополнительными параметрами, и так далее. Используя «телескопический конструктор», становится трудно писать код клиента, когда имеется много параметров, а еще труднее этот код читать. Читателю остается только гадать, что означают все эти значения и нужно тщательно высчитывать позицию параметра, чтобы выяснить, к какому полю он относится. Длинные последовательности параметров одного типа могут приводить к тонким ошибкам. Если клиент случайно перепутает два из таких параметров, то компиляция будет успешной, но программа будет работать не верно.
2) Паттерн ***JavaBeans*** – Вы вызываете конструктор без параметров, чтобы создать объект, а затем вызываете сеттеры или используете публичные свойства для установки обязательных и дополнительных параметров, представляющих интерес. Паттерн ***JavaBeans*** не лишен серьезных недостатков. Поскольку строительство разделено между несколькими вызовами, ***JavaBean*** может находиться в неустойчивом состоянии,

частично пройдя через конструирование. Попытка использования объекта, если он находится в неустойчивом состоянии, может привести к ошибкам, которые далеки от кода, содержащего ошибку, и, следовательно, трудными для отладки. Также ***JavaBeans*** паттерн исключает возможность сделать класс неизменным(immutable), что требует дополнительных усилий со стороны программиста для обеспечения безопасности в многопоточной среде.

Решением в подобной ситуации может быть паттерн ***Fluent Builder***, который позволяет упростить процесс создания сложных объектов с помощью методов-цепочек, которые наделяют объект каким-то определенным качеством. Применение данного паттерна делает процесс конструирования объектов более прозрачным, а код – более читабельным.

Вместо непосредственного создания желаемого объекта, клиент вызывает конструктор (или статическую фабрику) со всеми необходимыми параметрами и получает объект строителя. Затем клиент вызывает сеттер-подобные методы у объекта строителя для установки каждого дополнительного параметра. Наконец, клиент вызывает метод Build() для генерации объекта, который будет являться неизменным(immutable).

Задание

На основе паттерна ***Fluent Builder*** реализовать класс EmployeeBuilder, позволяющий строить резюме соискателя.

Рекомендации по выполнению:

1. В класс Employee рекомендуется добавить метод CreateBuilder() со следующей сигнатурой:

```
public static EmployeeBuilder CreateBuilder()
    {
        return new EmployeeBuilder();
    }
```

который возвращает объект строителя.

2. В классе строителя `EmployeeBuilder` можно дополнительно перегрузить операцию неявного приведения к типу `Employee`

```
    public static implicit operator Employee(EmployeeBuilder
builder)
        {
            return employeeBuilder.employee;
        }
```

что избавит от необходимости вызывать метод Build() для строительства объекта-соискателя.