

Отчет по практикуму: Моделирование работы страховой компании

13 ноября 2025 г.

Содержание

1	Формулировка задания	2
2	Диаграмма классов	3
3	Спецификация основных классов	3
3.1	Policy	3
3.2	HousePolicy / CarPolicy / HealthPolicy	4
3.3	Company	4
4	Диаграмма объектов	4
5	Использованные инструментальные средства	4
6	Модульная структура программы	5
7	Описание пользовательского интерфейса	5

1 Формулировка задания

Задача заключалась в разработке модели управления страховой компанией, выполняющей страхование населения по трём направлениям:

- Страхование жилища
- Страхование автомобиля
- Страхование здоровья

Цель симуляции — дать пользователю (менеджеру компании) возможность управлять условиями страховых продуктов и наблюдать за финансовым состоянием компании в течение нескольких месяцев. Модель включает:

- Продажу полисов в зависимости от базового спроса и параметров продукта
- Выплаты по страховым случаям
- Уплату налога
- Возможность обновления условий полисов
- Риск банкротства компании

2 Диаграмма классов

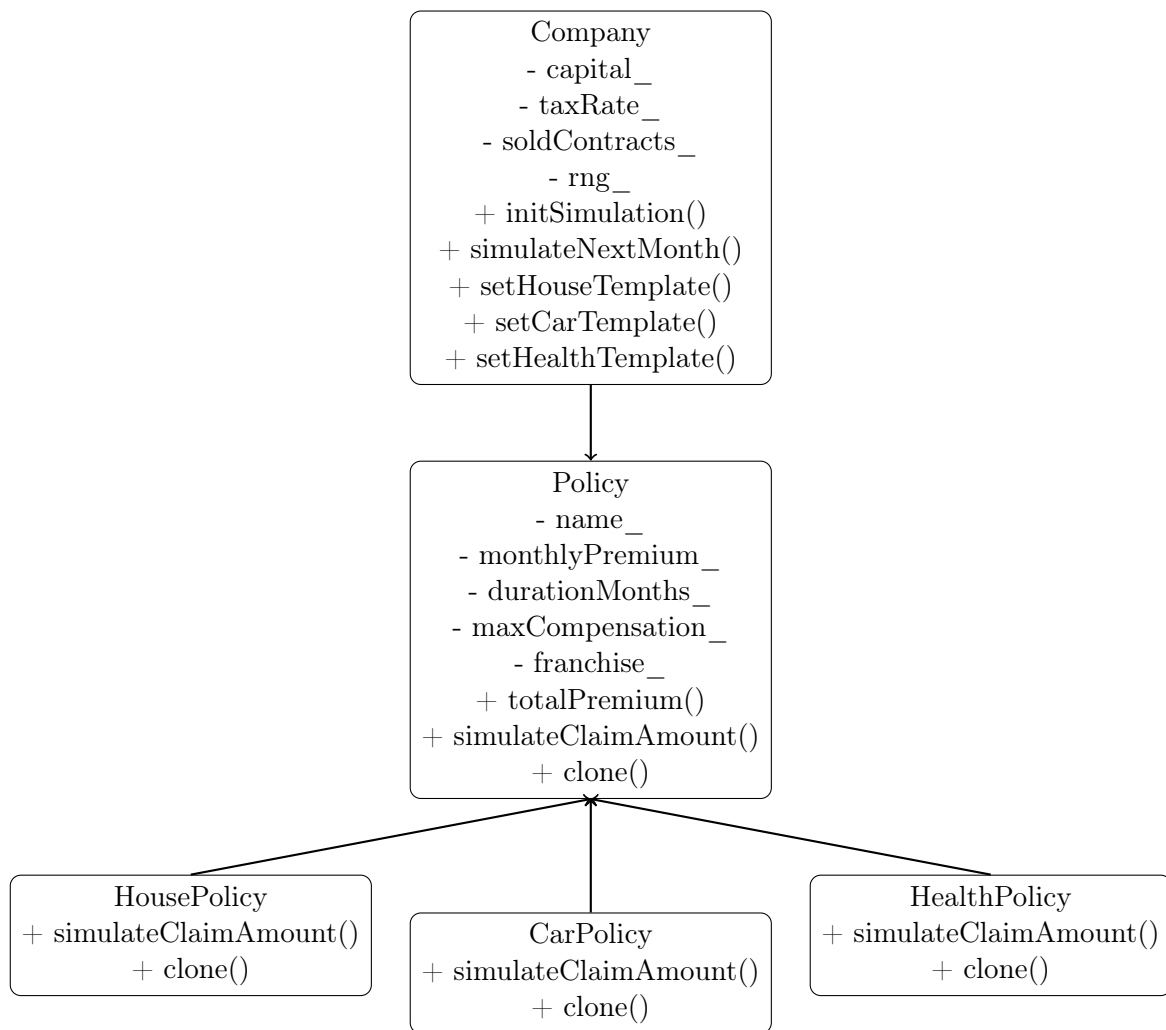


Рис. 1: Диаграмма классов программы

3 Спецификация основных классов

3.1 Policy

```
1 // Base class for all policies
2 class Policy {
3 public:
4     Policy(const QString& name, double monthlyPremium, int
5           durationMonths,
6             double maxCompensation, double franchise);
7     virtual ~Policy();
8     QString name() const;
9     double monthlyPremium() const;
10    int durationMonths() const;
11    double maxCompensation() const;
12    double franchise() const;
13    virtual double totalPremium() const;
14    virtual double simulateClaimAmount(std::mt19937 &rng) const = 0;
```

```

14     virtual std::unique_ptr<Policy> clone() const = 0;
15 };

```

3.2 HousePolicy / CarPolicy / HealthPolicy

Наследники класса Policy, реализуют метод `simulateClaimAmount()` для имитации страховых случаев.

3.3 Company

```

1  class Company {
2  public:
3      Company(double initialCapital = 30000.0, double taxRate = 0.09);
4      void setBaseDemandHouse(int d);
5      void setBaseDemandCar(int d);
6      void setBaseDemandHealth(int d);
7      void setHouseTemplate(std::unique_ptr<Policy> p);
8      void setCarTemplate(std::unique_ptr<Policy> p);
9      void setHealthTemplate(std::unique_ptr<Policy> p);
10     void setTrustFactor(double factor);
11     void initSimulation(int monthsToSimulate, std::uint32_t rngSeed =
12         std::random_device{}());
13     QString simulateNextMonth();
14     double capital() const;
15     bool isBankrupt() const;
16     int currentMonth() const;
17 private:
18     double capital_;
19     double taxRate_;
20     std::vector<SoldContract> soldContracts_;
21     std::mt19937 rng_;
22 };

```

4 Диаграмма объектов

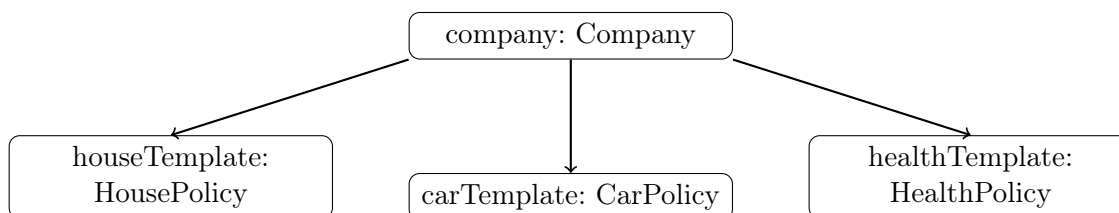


Рис. 2: Диаграмма объектов программы

5 Используемые инструментальные средства

- Язык C++17
- Qt6 Widgets (QMainWindow, QPushButton, QLineEdit, QTextEdit, QLabel, QTableWidget)

- Стандартная библиотека C++ (std::vector, std::unique_ptr, std::mt19937, std::uniform_int_distribution, std::uniform_real_distribution)
- CMake 3.16+

6 Модульная структура программы

- **policy.h / policy.cpp** — классы Policy и наследники, фабричные функции создания полисов
- **company.h / company.cpp** — класс Company, управление капиталом, продажами и выплатами
- **mainwindow.h / mainwindow.cpp** — графический интерфейс, управление симуляцией
- **main.cpp** — точка входа приложения
- **CMakeLists.txt** — сборка проекта

7 Описание пользовательского интерфейса

Программа содержит главное окно с тремя основными блоками:

1. **Параметры компании:** ввод начального капитала, налоговой ставки, количества месяцев моделирования.
2. **Условия полисов:** ввод ежемесячного взноса, длительности, максимальной суммы возмещения и франшизы для каждого типа полиса.
3. **Базовый спрос и доверие:** настройка базового спроса по каждому типу полиса и коэффициента доверия населения.

Ниже расположены кнопки управления:

- Инициализация симуляции
- Следующий месяц
- Обновить условия полисов

В нижней части окна отображается журнал симуляции и текущий капитал компании.