**Описание программы**

Данная программа представляет систему, позволяющую:

* хранить и управлять различными видами нагревательных приборов (SealedHeater, UnsealedHeater, Tub\_el\_Heater, SpiralHeater);
* добавлять приборы вручную или загружать их из файла;
* сохранять информацию в файл;
* выполнять поиск по имени, цене, мощности;
* сортировать по цене;
* работать с матрицей приборов и списком (на базе связного списка).

**Структура и логика программы**

**Основные классы:**

* **IHeater** — абстрактный интерфейс, определяющий базовые методы для всех типов нагревателей.
* **Heater** — абстрактный базовый класс, реализующий общие свойства: имя, цвет, цена, мощность.
* **Производные классы**:
  + SealedHeater — герметичный нагреватель с температурой;
  + UnsealedHeater — негерметичный нагреватель с длиной;
  + Tub\_el\_Heater — трубчатый нагреватель с толщиной стенок;
  + SpiralHeater — спиральный нагреватель с флагом "удлинён".

**Дополнительные структуры:**

* **TemplateMtrx<T>** — шаблонная матрица приборов.
* **Lister<T>** — шаблонный список приборов, реализованный на основе VoidLister.
* **VoidLister** — универсальный однонаправленный связный список на void-указателях.

**Объектно-Ориентированное описание**

**1. Инкапсуляция**

Все поля в классах объявлены как private или protected, доступ к ним осуществляется через **геттеры/сеттеры**, что защищает внутреннее состояние объектов.

Пример:

void Heater::setprice(int newprice) {

if (newprice < 0) throw out\_of\_range("Error price\n");

price = newprice;

}

**2. Наследование**

* Абстрактный интерфейс IHeater задаёт **контракт** для всех типов нагревателей.
* Класс Heater реализует общие характеристики и поведение.
* Конкретные нагреватели (Sealed, Unsealed и т.д.) **наследуют** Heater и реализуют свои особенности.

Иерархия:

IHeater

↑

Heater

↑ ↑ ↑

Sealed Unsealed Tub\_el

↑

Spiral

**3. Полиморфизм**

Методы print, sfile, print\_info, print\_header — **виртуальные**, что позволяет работать с объектами разных типов через указатель IHeater\*.

void Heater::print() const {

print\_header(); // виртуальный

...

print\_info(); // виртуальный

}

Таким образом можно, например, вызвать print() у IHeater\*, и будет вызван правильный метод для SpiralHeater, Tub\_el\_Heater и т.д.

**4. Абстракция**

Ты абстрагировал общие свойства нагревателей в базовый класс и интерфейс, чтобы пользователю не нужно было знать детали реализации каждого типа.

**Примеры применения принципов SOLID**

* **S (Single Responsibility)** — каждый класс отвечает только за свою задачу: Heater — за общие свойства, TemplateMtrx — за хранение, Lister — за управление списком.
* **O (Open/Closed)** — система легко расширяется: можно добавить новый тип нагревателя без изменения существующих классов.
* **L (Liskov Substitution)** — можно использовать любой Heater вместо IHeater, и всё будет работать.
* **I (Interface Segregation)** — интерфейс IHeater не перегружен, содержит только необходимые методы.
* **D (Dependency Inversion)** — в Lister и Matrix ты используешь шаблоны, что позволяет подставлять любые классы.