**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Математическая кибернетика и информационные технологии»

**Отчет по лабораторной работе №2**

по дисциплине «Информационные технологии и программирование»

Выполнил: студент группы

БПИ2401

Старков Дмитрий Константинович

Проверил:

Харрасов Камиль Раисович

Москва, 2025

Содержание

[Цель работы 2](#_Toc210398682)

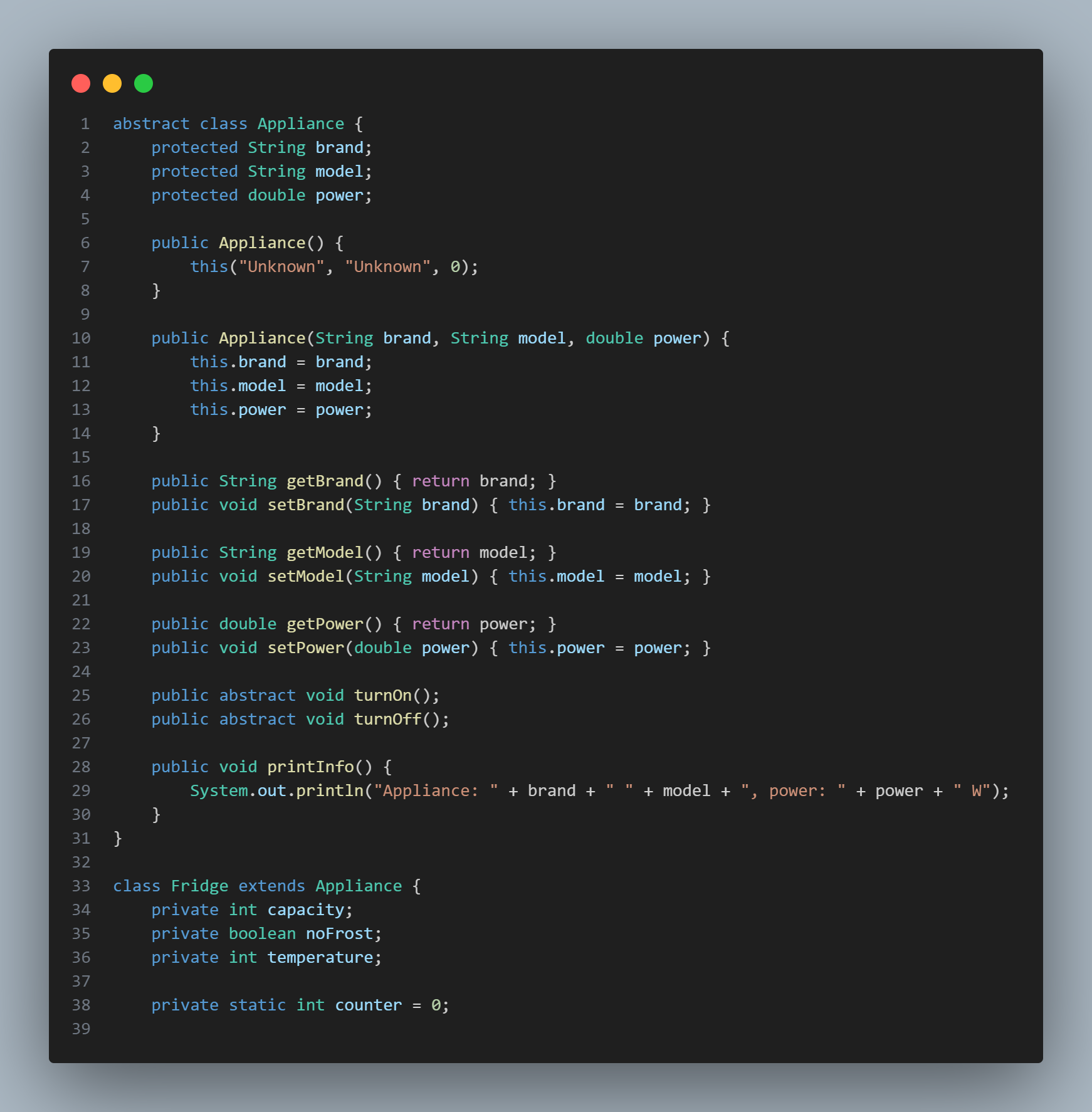
[Ход работы 2](#_Toc210398683)

[Вывод 7](#_Toc210398684)

Цель работы

Освоить принципы объектно-ориентированного программирования (ООП).

Ход работы

Начнем выполнение данной лабораторной с создания общего родительского абстрактного класса Appliance, вот как будет выглядеть код:

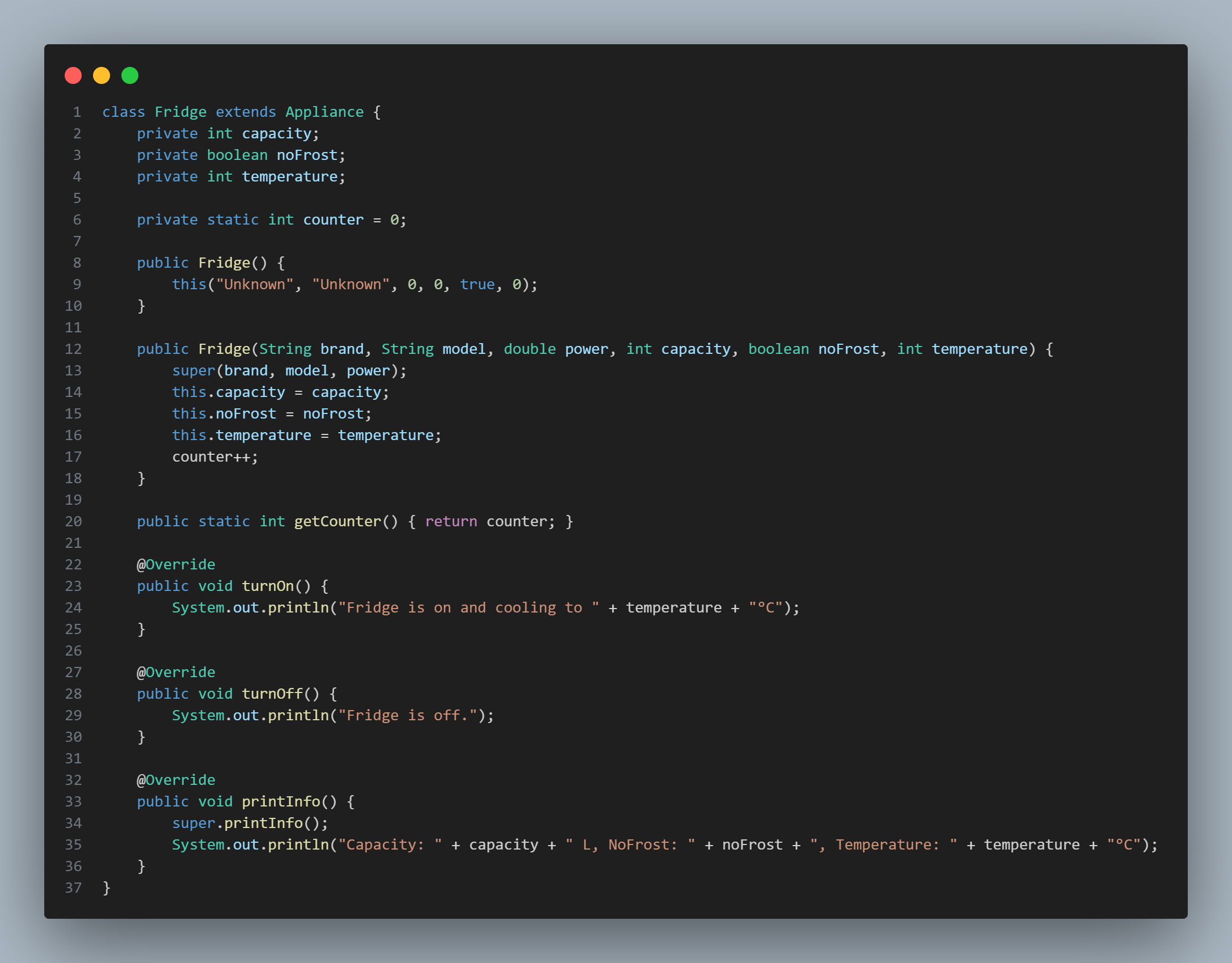
В начале инициализирую 3 поля, которые будут у всех подклассов – бренд, модель и мощность, эти 3 поля я делаю типа Protected, чтобы дочерние классы имели к ним прямой доступ, это конечно нарушает строгую концепцию инкапсуляции, но зато мне будет удобнее работать с ними в дочерних классах.

Далее идет конструктор, который вызывается при отсутствии аргументов в объявлении нового объекта данного класса. В этом конструкторе через оператор this я устанавливаю занчения Unknown и 0.

Джава понимает, какое значение относится к каждому атрибуту, потому что ниже у меня еще один конструктор, в котором переменные совпадают с типом значений.

Далее у меня идёт объявление сеттеров и геттеров, а также абстрактных классов turnOn и turnOff, которые в последствии будут перезаписаны для каждого дочернего класса. Ну и в конце общий метод printInfo, который будет выводить общую информацию для бытовой техники.

Переходим к первому дочернему классу – Холодильник:



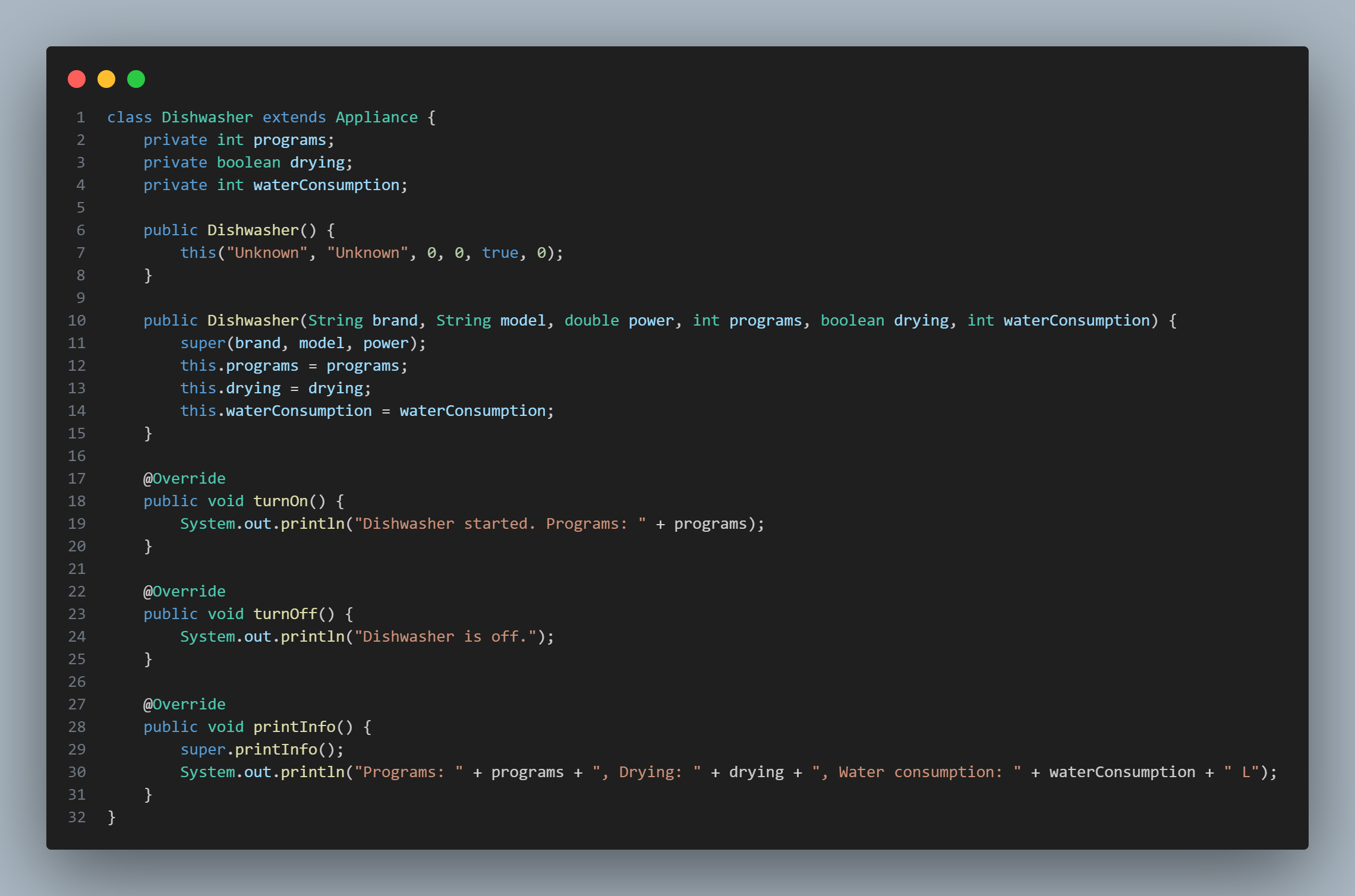
Здесь я инициализирую уже 3 приватных поля с объемом холодильника, наличием/отсутствием морозильной камеры и его температурой. Далее идёт объявление счетчика, который будет отображать количество объектов данного класса.

После чего идут конструкторы. Первый из них создан для пустого вызова, и несет в себе значения по умолчанию. Второй конструктор уже для общего случая. Первой строкой идёт ссылка на конструктор для родительского класса с помощью оператора super, а далее уже пристваиваются значения для уникальных атрибутов данного класса.

Отмечу, что оператор this используется из-за одинакового названия атрибутов класса и переменных, переданных в конструктор. Это нужно для того, чтобы не происходило двусмысленности, из-за которой операция х=х расценивалась бы как присвание переменной х собственного значения. Оператор this позволяет точно отделить поле класса и переменную, которую нужно туда записать.

В конце конструктора идет инкрементация счетчика, означающая создание очередного объекта данного класса.

Далее идет определение метода getCounter(), чтобы получать значение счетчика. Ну и последнее – перезапись родительских методов c предварительной аннотацие @Override, которая однозначно определяет перезапись родительского метода, и в случае, если именя не совпадают – выдастся ошибка, которая предотвратит создание нежелательного метода у класса. Отмечу также использование super.printInfo() в последнем методе. Данную конструкцию я буду использовать во всех оставшихся классах, это позволит сначала вывести общую информацию о приборе, а потом уже более специфическую.

Следующуй класс:

Здесь особо объяснять нечего, ситуация аналогична с предыдущим классом. Идёи далее:



Вот в этом примере все как обычно, за исключением того, что я решил взять этот класс за основу для создания нового дочернего класса, именно поэтому, по аналогии с абстрактным общим классом Appliance, я сделал его поля protected, а не private. Теперь к самому дочернему классу:

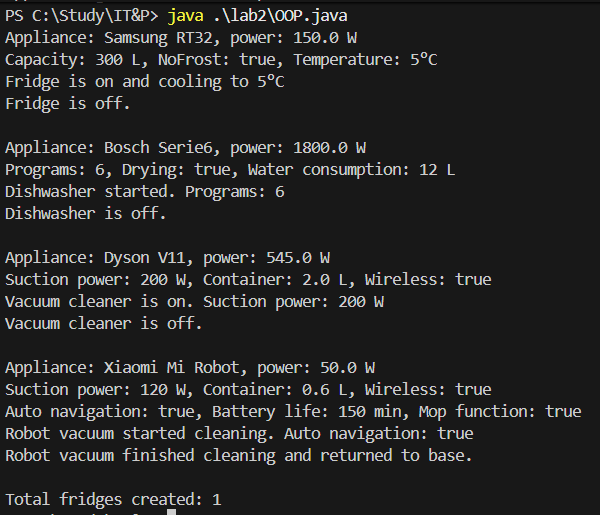


Тут тоже все как обычно, единственное отличие – большее количество полей ввиду двойного наследования, вышло целых 9 показателей, информация о которых будет выведена в 3 строки, так как super.printInfo() будет вызван как в этом, так и в родительском классе.

Итак, все классы готовы, осталось написать основной класс, создать пару объектов и вызвать каждый метод для демонстрации функционала:



Вот код, а вот итоговый вывод:



Вывод

В ходе лабораторной работы мной были освоены все 4 принципа ООП.

Ссылка на гит - https://github.com/BestStarProggramer/IT-P/tree/main/lab2