ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра прикладної математики
(повна назва кафедри)

МАГІСТЕРСЬКА РОБОТА

МОДЕЛЮВАННЯ СИСТЕМИ «РОЗУМНИЙ ДІМ» ІЗ ВИКОРИСТАННЯМ ШАБЛОНІВ ПРОЄКТУВАННЯ

Студентки 2 курсу, групи ПМПм-22с,
напряму підготовки Прикладна математика
Крохіної Є. М.
(прізвище та ініціали)

Керівник Дяконюк Л. М.
(прізвище та ініціали)

Рецензент

Національна шкала _____
Кількість балів: _____Оцінка: ECTS _____

Львів – 2023

Анотація

Ринок технології «розумний дім» досить добре розвинений, але не завжди враховує запит клієнтів, яким потрібні індивідуальні рішення. З метою створення продукту для таких користувачів була розроблена програма з використанням шаблонів проєктування. Вона дозволяє тестувати різні позиції датчиків і різні сценарії роботи контролерів. Програму було розроблено мовою Java у середовищі Eclipse. У роботі описані шаблони проєктування, що були реалізовані.

Ключові слова: шаблони проектування, контролери, розумний будинок, моделювання, поверховий план.

Abstract

The Smart home market is well-developed but fails to consider customers who have a need to create customized solutions. With the purpose to create a product for such users, a program using design patterns was developed. It allows testing different positioning of sensors and various scenarios of controllers’ work. The program has been developed using Java and Eclipse IDE. Design patterns that had been implemented, are described in this paper.

Keywords: design patterns, controllers, smart house, simulation, floor plan.

Зміст

Вступ

Із кожним роком людство впроваджує усе більше високотехнологічних рішень у повсякденне життя.

Однією із спроб покласти виконання рутинних обов'язків на автоматизовані рішення є «Розумний дім» - координована система датчиків та сенсорів, контрольованих програмним забезпеченням, яка виконує завдання із забезпечення комфорту та безпеки життя мешканців. Основною особливістю такої системи є можливість під'єднати усі пристрої в одну систему, яка має можливість віддаленого керування та отримати «екосистему», в якій різні аспекти працюють паралельно, але водночас є з'єднаними в єдине ціле за допомогою центрального контролера. Делегування рутинних обов'язків контрольованим системам дозволяє мешканцям більш продуктивно витрачати свій час.

Індустрія «розумних будинків» швидко розвивається, та за оцінками, зазначеними в [2], матиме зростання міжнародного ринку до сімдесяти п'яти мільярдів доларів. Цьому сприяє популяризація інноваційних рішень, але найбільшим чинником зростання є здешевлення приладів, що дозволяє значно розширити сегмент користувачів. У лідерах цього ринку є США та Китай, але в Україні, принаймні до повномасштабного вторгнення, ця індустрія також розвивалась[4].

Одними із найпоширеніших функцій «Розумного будинку» згідно із [3] є економія електроенергії, і цей аспект було розглянуто у одній із моїх попередніх робіт [1], а також безпека будинку, що буде основним аспектом, який буде розглядатись у цій праці.

На даний час в Україні поширеним варіантом «Розумного будинку» є готове рішення від забудовників, яке включає в себе базові функції, а розширення їх здійснюється вже самим власником виходячи з його запитів і фінансових можливостей. [5] Також є компанії, які пропонують встановлення подібного типу систем у готовому приміщенні проте є сегмент покупців, які хотіли би індивідуального підходу із проектування системи, маючи можливість підібрати

набір сенсорів та датчиків так, щоб це якнайбільше відповідало їхнім потребам, і саме для таких користувачів було розроблено систему із моделювання роботи «розумного дому» у цій та попередніх роботах[1][9].

Серед причин, що зупиняють активний розвиток ринку «Розумних будинків» в Україні є також наступні фактори, що описані в [6]:

- недовіра до рівня безпеки персональних даних(і небезпідставно, адже навіть великі компанії, як Фейсбук та Тесла використовують приватні дані у власних цілях[7])
- недостатня кількість коштів у більшості населення
- складна ситуація на ринку праці(із рівнем безробіття у 30% згідно з [8]).

Тому доцільною є розробка програми, яка би дозволила користувачам самостійно спроектувати поверховий план власного(чи майбутнього) помешкання, розмістити різні датчики та протестувати різноманітні сценарії їхньої роботи. А потім, скориставшись змодельованою системою, користувач може самостійно встановити датчики, адже на сьогоднішній день електроніка(мікроконтролери, сенсори, датчики) є доступними для будь-якого бюджету, починаючи від італійських Arduino uno до їх бюджетних аналогів із Aliexpress, що дозволяє обладнати помешкання незалежно від рівня доходу.[1]

Для розробки даної програми буде використано шаблони проєктування. Вони дозволяють зробити програму більш універсальною: забезпечують можливість повторного використання коду програми для суміжних проєктів та розширення спектра функцій поточного проєкту.

Розділ 1.
Огляд предметної області

1.1 Розумний дім.

Розумний дім – ідея, що зародилась досить давно. У книжках науково-фантастичного спрямування автори описували будинки, де усі процеси виконують роботи, та й з плином прогресу людство завжди намагалось перекласти частину своїх обов'язків на автоматизовані рішення.

Перші розумні будинки були радше набором ідей, роздумами з автоматизації побуту[12]. Проте із виникненням, розвитком та популяризацією сучасних технологій, з'являлося все більше засобів, що допомагають втілити ці ідеї в життя.

Першою спробою з автоматизації можна назвати пульт керування Ніколи Тесли ще у 1898 році, а першим «розумним» приладом вважають порохотяг, після винайдення якого спостерігався справжній «бум» появи різноманітних приладів для допомоги із хатніми справами. І хоч їх не можна вважати «розумними приладами» у сучасному трактуванні терміна, проте це були початкові спроби із впровадження технологій у побут людських осель. [13] Найпершою ж інтегрованою системою керування будинком, яка мала можливість керування через комп'ютер, була ECHO IV[14], яка хоч і була інноваційною, проте так і не здобула популярності через високу вартість на тогочасний 1966ий рік.

До цієї технології людство знову повернулось після зниження вартості електроніки. Збільшення потенційних користувачів призвело до появи різноманітних нових пристроїв та систем.

Зараз досить велика частина споживачів обирають саме «розумні» прилади для свого дому: одні купують окремі пристрої, інші – цілі системи.

До того ж, дослідження[15] показують, що на теперішній час серед людей, що обирають «розумні» пристрої є не лише власники нових помешкань, а й ті, хто мають бажання удосконалити власне житло.

Основними причинами росту ринку «розумних будинків» згідно з [15] є:

- Простота встановлення та поєднання «розумних» пристроїв
- Безпека від проникнення та витоків(води, газу)
- Економія електроенергії(позитивний вплив і з погляду економії грошових ресурсів, і з огляду на вплив на екологічну ситуацію)

Пандемія коронавірусу посприяла ще більшому росту ринку «розумних будинків». По-перше, через величезний ріст відсотка людей, що працюють чи навчаються вдома, по-друге, через те, що багато людей залишились без роботи і вирішили втілювати давні бізнес-ідеї в життя. Згідно з [17] до 2026 року прогнозовано ріст ринку до 138.9 мільярдів доларів США, індустрія «розумних будинків» є одною із найдинамічніших. Країнами-лідерами впровадження «розумних будинків» є США, Китай та деякі країни Європи.

Є безліч варіацій використання контрольованих систем – це і керування побутовими приладами, і імітування присутності мешканців вдома за допомогою вмикання світла у кімнатах, вибраних випадковим чином, щоб запобігти крадіжкам. Але найпростіші, і тому найпоширеніші системи зосереджені на регулюванні освітлення, опалення, кондиціонування приміщень та вентиляції. [1]

Сучасні тенденції з автоматизації будинків – це автоматичне регулювання термостата для збільшення чи зменшення температури, регулювання освітлення, засоби відеоспостереження для захисту будинку, а також отримання сповіщень про зміни в системі чи виконану роботу[16].

«Розумний будинок» створюється шляхом об’єднання систем освітлення, опалення, охолодження, охорони в єдину керовану систему. Така автоматизація призводить до підвищення комфорту людського життя, а також сприяє підвищенню енергоефективності. [29]

Також однією із найважливіших причин, чому люди обирають «розумний будинок» - це додаткова безпека. Як видно на рисунку 1, спостереження та безпека будинку – це один із найбільших сегментів ринку.

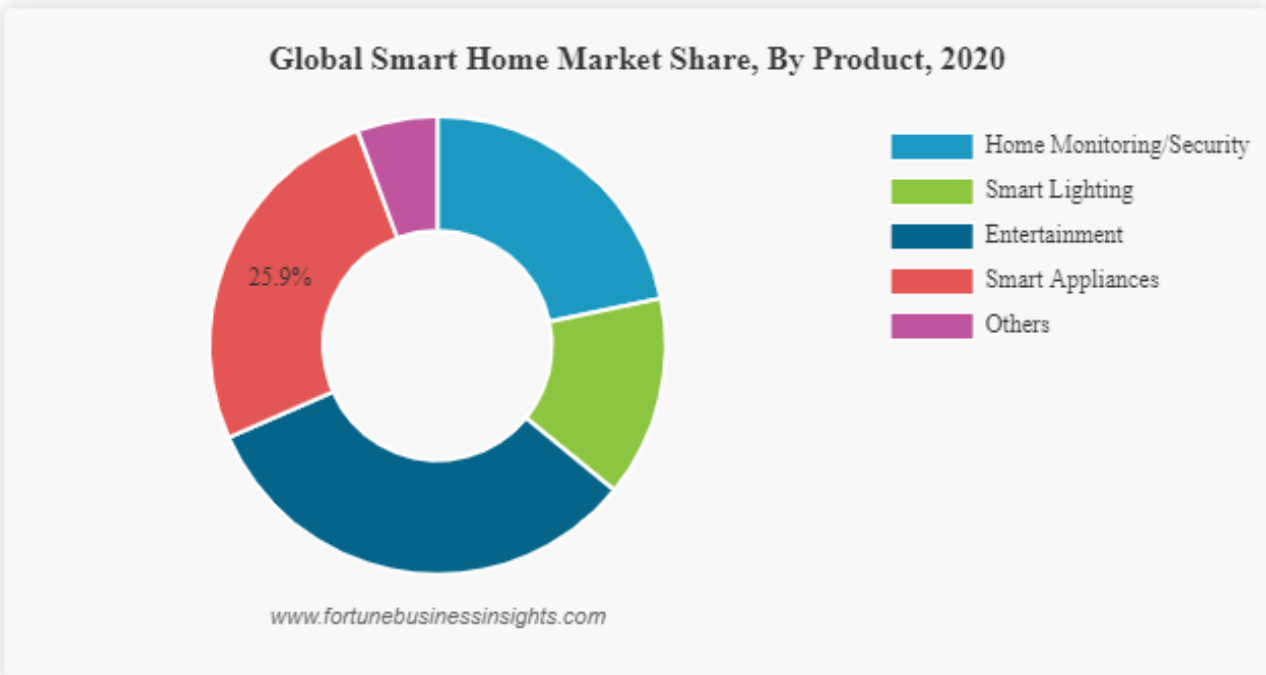

Рис. 1 Сегменти ринку «розумних будинків»[19].

Сучасні системи «розумних будинків» зосереджені на безпеці мешканців та переході до більш «зеленого» способу життя і варто зазначити, що за прогнозами, наведеними на рисунку 2, сектор безпеки в індустрії «розумних будинків» лише зростатиме, і вже у 2026 році очікується збільшення ринку утричі.

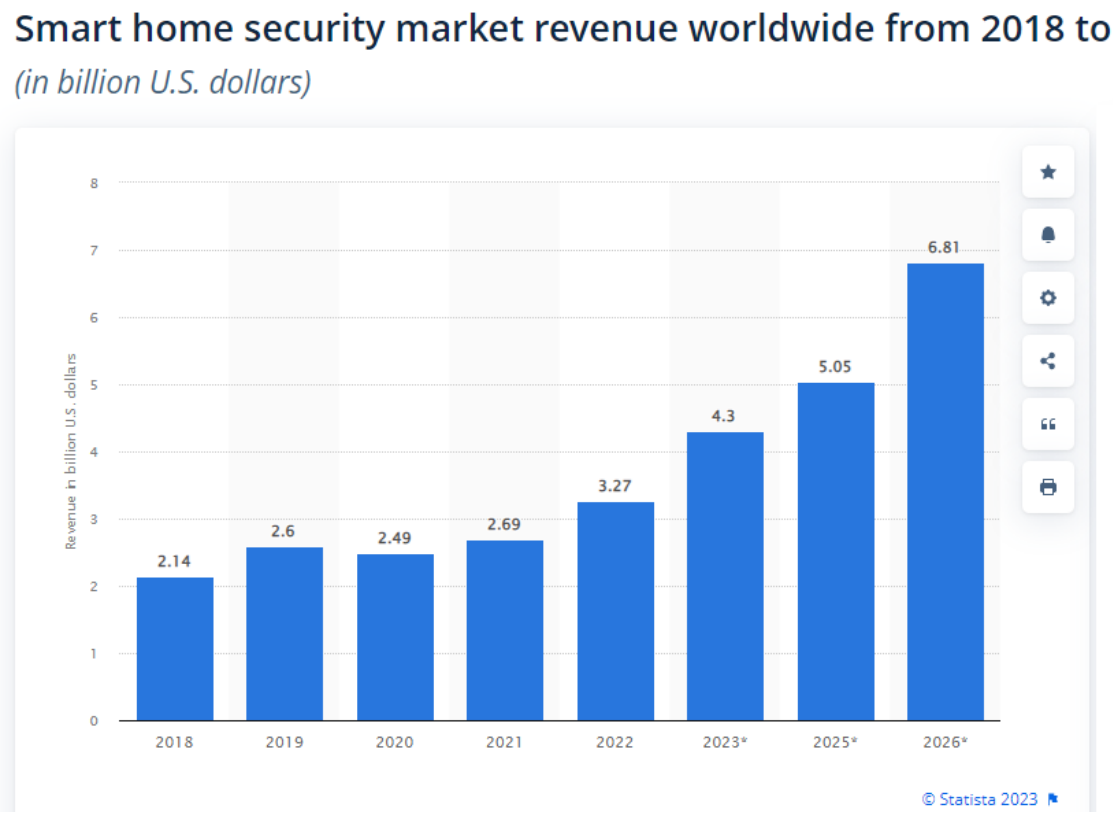

Рис. 2 Прогноз зростання ринку безпекового сектора «розумних будинків»[18]

Також варто зазначити, що, як видно на рисунку 3, впровадження системи «розумного будинку» призводить до зменшення використання електроенергії, що на сьогоднішній день є не лише питанням економії, що вже було розглянуто у [1], а й питанням енергетичної безпеки України, адже через війну багато вітчизняних потужностей, що виробляють електроенергію, або пошкоджені, або перебувають під тимчасовою окупацією.

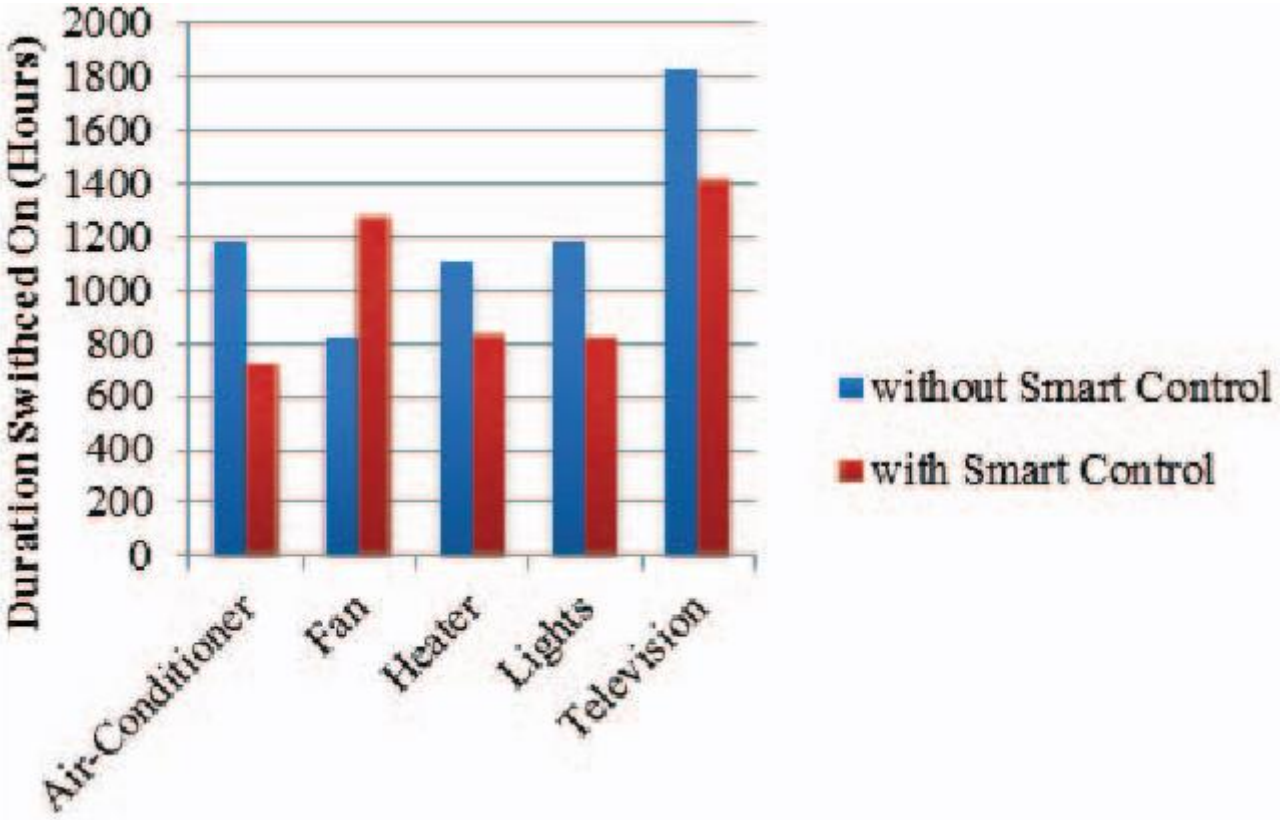

Рис. 3 Порівняння середньостатистичного часу роботи побутових приладів з та без системи «розумного контролю». [30]

1.2 Виникнення та розробка концепту шаблонів проєктування

Однією з перших спроб об’єднати та систематизувати рішення типових проблем у набір інструкцій була книга «Мова шаблонів: міста, будівлі, будівництво» авторства Крістофера Александера та інших. В ній були описані понад двісті п’ятдесят шаблонів, за допомогою яких можна було вирішувати будь-яку проблему проєктування, незалежно від її розміру, адже автор помітив, що для розв’язання будь-якого завдання архітектури, існує скінченний набір рішень.[10]

Цей концепт перейняли та популяризували майже два десятиліття потому чотири програмісти у книзі «Шаблони проєктування: Елементи об’єктно-орієнтованого програмного забезпечення, які можна повторно використати» для розробки програмного забезпечення . У ній були перелічені основні шаблони проєктування, які з часом стали класичними прийомами. Цей «прийом» став популярним у різних областях програмування, тому поняття шаблонів проєктування зустрічається і за межами об’єктно-зорієнтованого проектування.[26]

Шаблон проєктування – типовий спосіб вирішення певної проблеми, яка часто зустрічається при проєктуванні. На відміну від готових функцій та бібліотек, шаблони не є готовими частинами коду, а лише містять інформацію про підхід до вирішення проблеми. [11] Це робить їх універсальним підходом до рішення проблеми, адже їх можна зреалізувати будь-якою мовою програмування. Шаблони - це сценарії, за допомогою яких можна вирішити проблеми, що виникають при розробці, але їх потрібно реалізовувати згідно із потребами конкретної програми.

1.3 Оцінка якості програмного забезпечення

Якість програмного забезпечення – один із найважливіших аспектів розробки. При розробці програмного забезпечення важливим є фактор повторного використання коду, що економить грошові та людські ресурси. Програмне забезпечення хорошої якості пришвидшує майбутню розробку та оновлення, а також знижує вартість розробки та обслуговування.

Якісним буде вважатись той код, який легко підтримувати, є надійним, має достатній рівень безпеки, є достатньо оптимізованим та стабільним, також він має добре та швидко працювати, а також продукт має бути легким у користуванні та зрозумілим різним категоріям користувачів.

Погано спроєктоване програмне забезпечення створює проблеми у його підтримці, а також такий код набагато дорожче підтримувати. На відміну від помилок виконання, помилки при проектуванні важко виявити за допомогою написання тестів. Тому ще на етапі проектування важливо оцінити всі нюанси роботи проєкту та витратити час та зусилля на продуманий дизайн.

Хороша підтримуваність коду забезпечує легкість впровадження нових функцій, оновлення версій, покращень коду, оптимізації та усунення недоліків. Такий код легко адаптувати не лише для нових версій, а й під нові потреби. Також якісно написаний код краще придатний для розуміння новим членам команди розробників, що спрощує водночас і комунікацію, і виконання поставлених задач, а також сприяє прогресу у розробці проєктів.

Підтримка коду є найбільш коштовною частиною будь-якого проєкту і зазвичай становить понад 50% від загального бюджету[21], тому саме проєктування є найважливішим етапом розробки. Також набагато дешевше здійснювати оновлення добре спроєктованого коду та додавати в нього новий функціонал. Тому якісно спроєктований продукт несе в собі «подвійну вигоду», яка компенсує час, витрачений на детальне, продумане проєктування.

Існує кілька способів визначення якості коду. Їх поділяють за способом аналізу на статичні та виконання. Спосіб виконання згідно з [22] оцінює такі характеристики:

- Середній час до моменту помилки. Вимірюється час між помилками, що виникли. Така метрика в основному використовується у тих системах, де час і точність виконання критично важливі: управління рухом літаків, зброя та авіоніка.

- Щільність виникнення помилок. Вимірюється відносна частота виникнення дефектів до загальної кількості коду(може вимірюватись як кількість стрічок коду, так і кількість функцій, методів). У більшості комерційних проєктів використовують саме цей принцип.

- Оцінка помилок із клієнтської точки зору. Вимірюється кількість помилок чи труднощів використання, що виникають у клієнта при користуванні. При такому способі не обирають конкретного клієнта, а обчислюють загальну кількість помилок, про які було повідомлено клієнтами за певний відрізок часу.

- Загальна оцінка користування клієнтом. Вимірюється середня оцінка із виставлених оцінок за користування клієнтами(зазвичай за п’ятибальною шкалою).

- Машинне тестування. Вимірює щільність появи дефектів та ефективність їх усунення.

- Оцінка виконання. Вимірюється час реакції відповідних елементів продукту – середнє значення виконання всіх процесів від моменту

запуску до завершення. Чим менший час реакції, тим кращим його вважатимуть користувачі.

До статичних метрик, тобто таких, що здійснюють вимірювання якості коду без виконання програми, згідно з [27] відносять:

- Цикломатична складність. Згідно з [23] такий спосіб оцінки вимірює кількість лінійно незалежних шляхів у програмі. Підраховуються усі входження в цикли, умовні та логічні оператори, винятки та інше. Чим нижчий цей показник, тим легшим код є для сприйняття та модифікації.

- Зв'язність класів. Зв'язність класів аналізує кількість класів, що використовуються у даному класі чи компоненті. Занадто тісні зв'язки між компонентами, залежність їх від конкретних класів, а не абстрактних інтерфейсів, зменшують гнучкість архітектури проєкту та перешкоджають повторному використанню[11]. Сильно зв'язаний код важко доповнювати чи змінювати, а також у такий код важко вносити нові елементи. Слабко зв'язані класи зменшують залежності між компонентами. Також слабка зв'язність дозволяє ділити програму на кілька менших логічно зв'язних частин, що спрощує їх підтримку. Згідно із [25] сильна зв'язність класів прямо впливає на програмні збої.

- Кількість стрічок коду. Багато проєктів є об'ємними, проте чим менше стрічок коду він буде містити, тим краще. Комерційні проекти не можуть складатися з десяти стрічок коду, проте варто оптимізовувати їхню кількість.

- Глибина наслідування. Стосується лише об'єктно-зорієнтованих мов програмування. Вимірюється максимальна кількість класів, від батьківського до того дочірнього, що вже не має нащадків. Добре вибудувана ієрархія наслідування є важливою, але потрібно пам'ятати

про те, що уся ланка батьківських класів впливатиме на кінцевий дочірній, адже йому потрібно буде зреалізувати усі батьківські методи. Також, за можливості, потрібно заміняти наслідування класів на реалізацію інтерфейсів, адже по-перше, це зменшує зв’язаність, а по-друге, практично усі мови не дозволяють множинного наслідування, тому клас, який вже має батьківський не зможе за потреби бути наслідуваним від ще одного класу.

- Індекс підтримки коду. Обчислюється за допомогою вищезгаданих цикломатичної складності, кількості стрічок коду, а також кількості операндів та операторів. Чим вищий цей індекс, тим легше підтримувати такий код. Згідно з [24] якщо код має індекс у межах від двадцяти до ста пунктів за стобальною шкалою, то такий продукт має високий індекс підтримки.

Згідно із дослідженнями, проведеними у [28], використання шаблонів проєктування позитивно впливає на підтримку коду та зменшує час виконання, що також свідчить про збільшення якості програмного продукту.

Розділ 2.
Деталі реалізації програми

2.1 Визначення ключових завдань програми

Система керування повинна бути достатньо простою у користуванні, щоб якомога більший прошарок користувачів зміг скористатись розробленим продуктом. Водночас програма повинна враховувати нюанси роботи різноманітних датчиків.[1]

Одним із недоліків системи «розумний будинок» є її вартість, а також те, що готові рішення не завжди відповідають потребам користувачів, адже існує безліч різноманітних пристроїв для «розумного будинку», що відрізняються функціональністю, діапазоном дії та ціною, але далеко не всі із переліку потрібні конкретному споживачеві, тому не має сенсу переплачувати за готові системи, коли можна змоделювати рішення під свої потреби. Тому було б доречно мати змогу користувачеві підібрати окремі рішення і серед розробок стартапів, і серед широкого спектра продукції на масовому ринку, відповідно до потреб та ціни, а також протестувати їхню роботу та взаємодію у конкретному приміщенні ще до покупки.

За основу буде взято програму, що була зреалізована в моїй попередній роботі [1], яка передбачає нанесення користувачем довільного поверхневого плану на канву, а також розміщення датчиків та сенсорів.

Окрім того, користувач сам може спроектувати поверховий план помешкання, розмістити необхідні датчики та обрати один із сценаріїв їх роботи. Програма дозволяє користувачеві обрати комфортний діапазон температури та вологості приміщень, зменшити споживання електроенергії та газу шляхом регулювання освітлення та опалення залежно від того, чи є хтось у кімнаті. Після тестування різних сценаріїв та різноманітного розміщення сенсорів користувач

може обрати потрібні пристрої та встановити їх самостійно згідно з висновками, отриманими після симуляції.

Найпоширенішими аспектами, з якими працюють системи «розумного будинку», як вже було зазначено у розділі 1, є системи оптимізації використання енергоресурсів, і цей аспект вже було опрацьовано у попередній роботі [1], а також безпека, спостереження та нагляд зі сповіщенням користувача про «небажаних гостей». Тому основним завданням при розробці було додавання безпекового фактору.

Варто зазначити, що при розробці програмного забезпечення важливим є фактор повторного використання коду, що економить грошові та людські ресурси. Тому розроблена програма окрім симуляції розумного будинку може також бути використана для проектування схематичних поверхових планів при проектуванні власного будинку або для демонстрації бажаного результату архітекторам чи будівельникам.

При розробці програми варто дотримуватись принципу єдиного обов’язку класів, щоб кожен клас мав одну конкретну задачу для його створення, код повинен залежати від абстракцій(інтерфейсів), а не від конкретної реалізації класів.

Також важливо віддавати перевагу композиції, а не успадкуванню[11]. Останнє варто використовувати лише за великої потреби, коли дочірній клас є розширенням попереднього, а не просто його модифікацією. До того ж Java, як і більшість мов, не дозволяє множинного наслідування класів, тому не варто використовувати наслідування без особливої потреби, адже якщо виникне потреба унаслідувати якийсь із системних класів, додати новий батьківський клас не буде змоги, і доведеться суттєво змінювати архітектуру вже спроєктованої системи.

Саме тому усі базові класи, які безпосередньо стосуються розробленої програми, базуються на інтерфейсах:

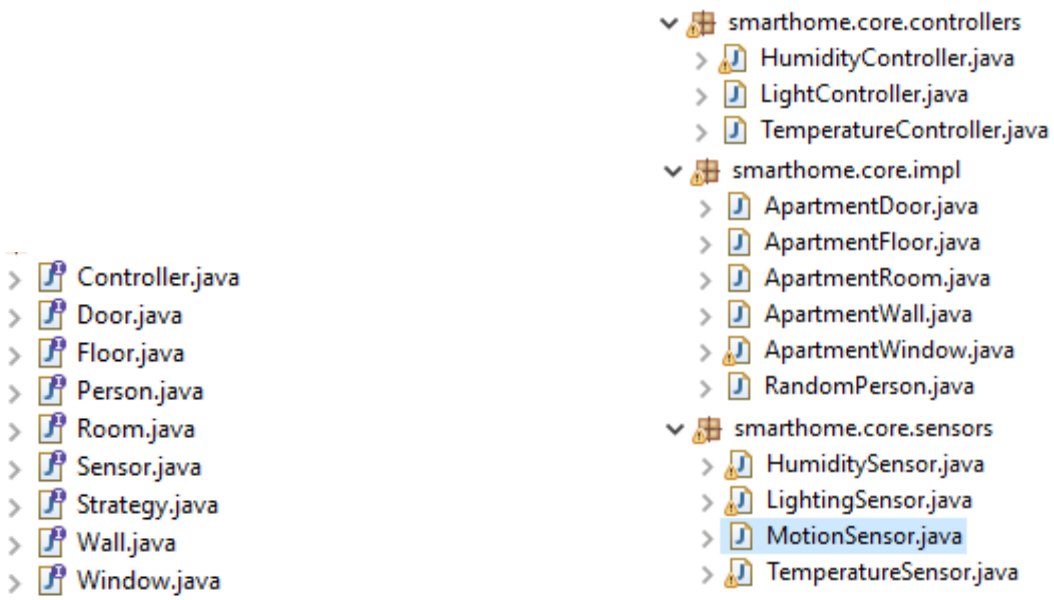

Рис. 4 Використання інтерфейсів у програмі

Код у класах залежить від інтерфейсів, тому у майбутньому при заміні одного класу, що реалізовує заданий інтерфейс на інший не буде викликати помилок.

Деякі допоміжні класи-утиліти, які не мають значного впливу на програму, та містять лише код, відокремлений за функціоналом від основних, та розширення яких у майбутньому не матиме сенсу, не було спроєктовано на базі інтерфейсів, адже це призвело би до ускладнення програми у більшій мірі, ніж імовірність розширення їх функціоналу у майбутньому.[1]

2.2 Шаблони проєктування

Шаблони проєктування спрощують повторне використання коду і його обслуговування і є перевіреними рішеннями для типових завдань, що виникають при розробці програмного забезпечення, тому можна стверджувати, що вони певною мірою впливають на якість коду.

Розрізняють три види шаблонів проектування[11]:

- породжувальні
- структурні
- поведінкові.

Породжувальні шаблони відповідають за створення об’єктів. Різні породжувальні шаблони містять сценарії створення різних об’єктів відповідно до конкретної ситуації, що виникає при розробці.

До породжувальних відносять такі шаблони[11]:

- Фабричний метод
- Абстрактна фабрика
- Будівельник
- Прототип
- Одинак

Структурні шаблони відповідають за побудову більших ієрархій класів, які зручно підтримувати.[11] Різні структурні шаблони забезпечують створення великих структур, визначення зв’язків між компонентами, їх успадкування та композицію. [20]

До структурних відносять такі шаблони[11]:

- Адаптер
- Міст
- Компонувальник
- Декоратор
- Фасад
- Легковаговик
- Замісник

Поведінкові шаблони описують взаємодію між об’єктами. Різні поведінкові шаблони забезпечують легку взаємодію між об’єктами, а також «слабку зв’язність» об’єктів – компоненти якнайменше залежать один від одного, явно не посилаються одне на одного. Наприклад, залежності формуються від інтерфейсів, а не конкретних класів.

До поведінкових відносять такі шаблони[11]:

- Ланцюжок обов'язків
- Команда
- Ітератор
- Посередник
- Знімок
- Спостерігач
- Стан
- Стратегія
- Шаблонний метод
- Відвідувач

Основними шаблонами, використаними у програмі, були: Одинак, Стратегія, Фасад, Спостерігач, Фабричний метод та інші.

2.2.1 Спостерігач

Спостерігач – це поведінковий шаблон проєктування, що дає змогу одним об’єктам стежити й реагувати на події, які відбуваються в інших об’єктах.[11]

Шаблон спостерігач використовується в тих випадках, коли потрібно певні об’єкти(підписники) повідомити про зміни чи події у інших об’єктах.

Цей шаблон пов’язує клієнта, видавця та інтерфейс підписника. Видавець зберігає посилання на об’єкти конкретних реалізацій підписника у колекції. Метод оновлення викликається автоматично, оскільки видавець знаходиться у стані «очікування», та за першої зміни у об’єкті, за яким він спостерігає, він «сповіщає» усіх підписників.

Підписники можуть підписуватись чи відписуватись і, оскільки їх список створюється динамічно[11], то процес підписки/відписки відбувається під час виконання програми.

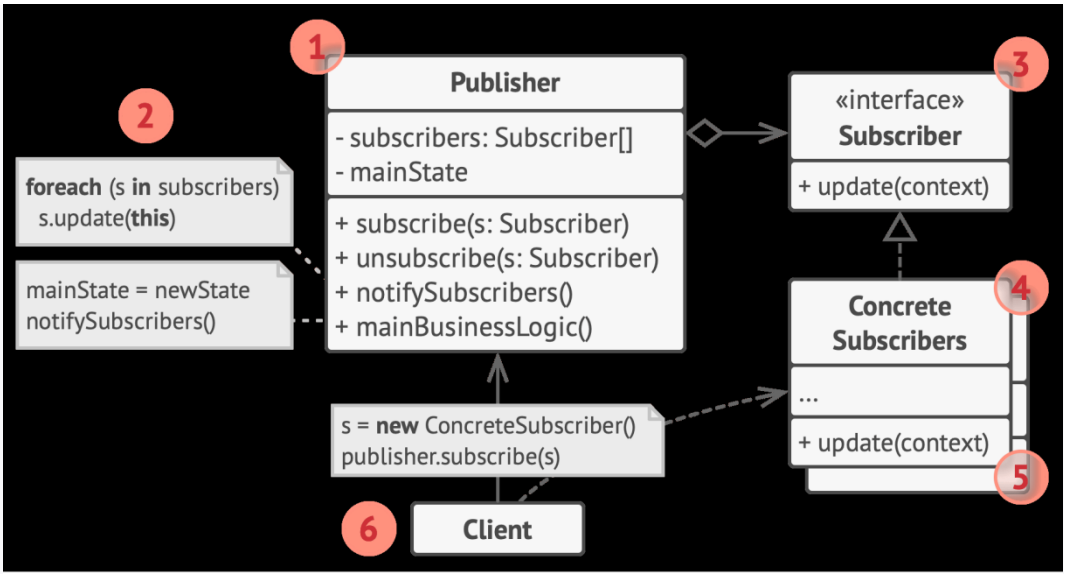

Рис. 5 Типова схема шаблону спостерігач

Цей шаблон було вибрано для реалізації безпекового фактору. Коли у кімнаті знаходиться сторонній об’єкт, користувач отримує сповіщення.

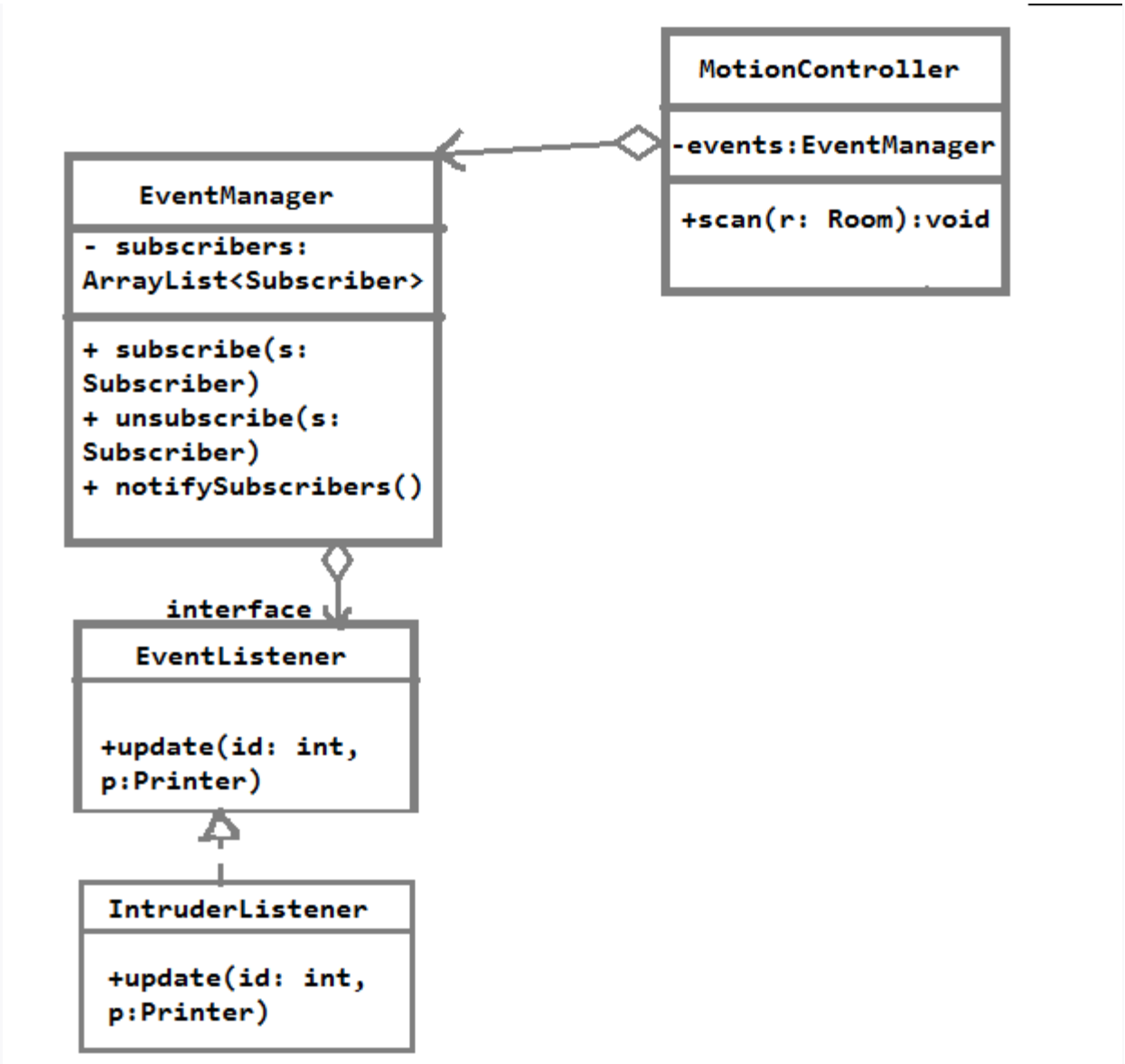

Рис. 6 Реалізація шаблону «Спостерігач»

EventManager відіграє роль видавця, EventListener – підписника. І у момент, коли у кімнаті помічено тепловий слід„ EventManager повідомляє усіх підписників про те, що з’явився хтось сторонній.

2.2.2 Посередник

Посередник — це поведінковий шаблон проектування, що дає змогу зменшити зв'язаність великої кількості класів між собою, завдяки переміщенню цих зв'язків до одного класу-посередника.[11]

Він забезпечує слабку зв'язність за допомогою інкапсуляції взаємодії об'єктів між собою. Це так-званий «зв'язківець», який бере на себе усю комунікацію. Тому елементи не є залежними один від одного, а існують паралельно і передають усі бажані дії посереднику, який вже обробляє інформацію та передає іншим об'єктам.

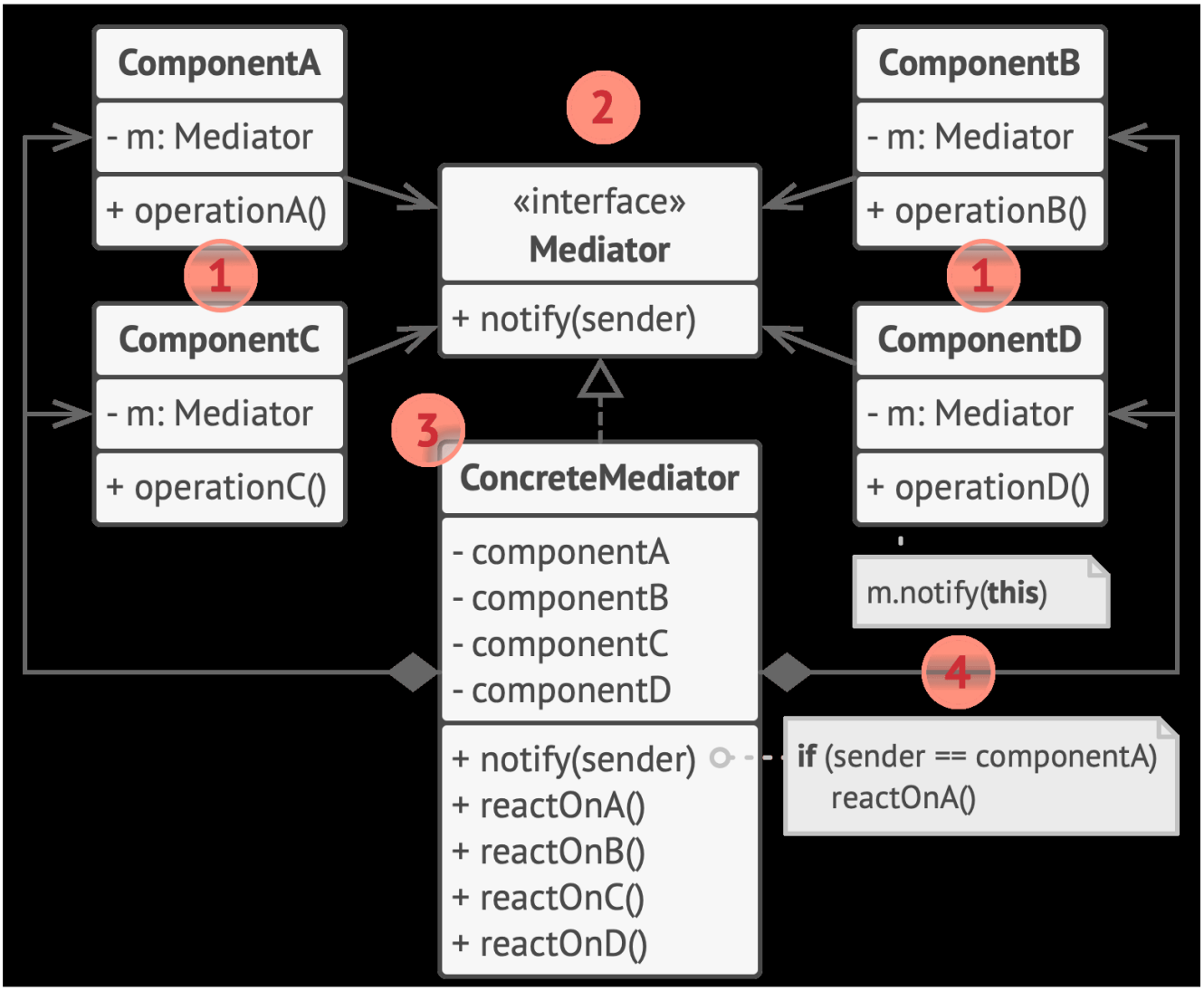

Рис. 6 Типова схема шаблону посередник

У розробленій програмі роль медіатора відіграють AppartmentManager та UIManager, що є реалізацями інтерфейсу.

2.2.3 Фабричний метод

Фабричний метод – це породжувальний шаблон проєктування, який визначає загальний інтерфейс для створення об’єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об’єктів[11].

Цей шаблон часто використовується, коли є різні способи виконання однієї і тої ж дії. І оскільки ці класи реалізовуватимуть спільний інтерфейс, їхні об’єкти можна заміняти у коді один на інший. Клієнт не відчує різниці між об’єктами різних класів, бо трактуватиме їх обох як представників того ж самого абстрактного інтерфейсу, які імплементують певний метод, який клієнт викликатиме, а конкретна реалізація методів у класах не впливає на працездатність коду клієнта.

У розробленій програмі таким чином розділено друк інформації про кімнати у консоль та у файл:

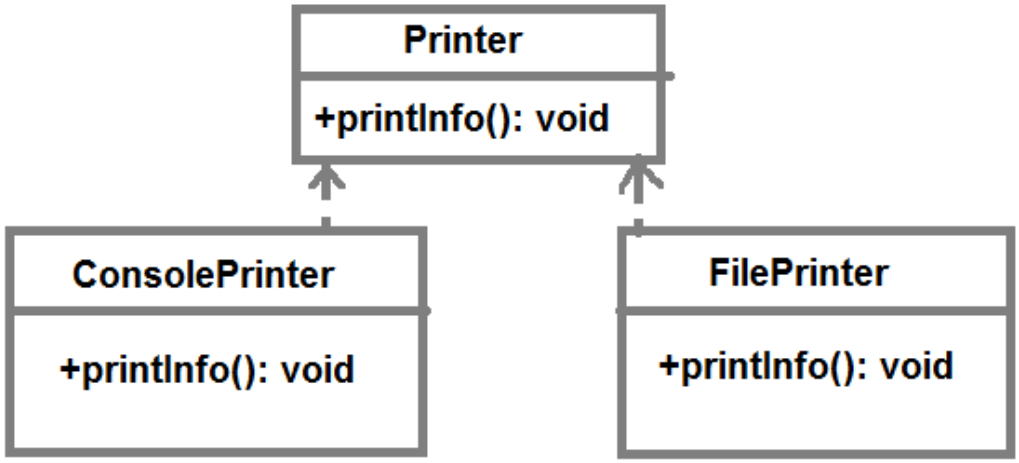

Рис. 7 Реалізація шаблону «Фабричний метод»

2.2.4 Стратегія

Стратегія – це поведінковий шаблон проєктування, який визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна заміняти один на інший прямо під час виконання програми. Стратегія дозволяє варіювати поведінку об'єкта під час виконання програми, підставляючи до нього різні об'єкти-поведінки[11].

Контролери діють як головний рушій усієї системи автоматизації. Вони аналізують інформацію, яку збирають із сенсорів та на основі цього приступають до виконання певного набору наперед визначених дій[12].

У програмі є можливість обрати варіант симуляції із випадного списку – одну зі стратегій поведінки контролерів. Залежно від вибраної стратегії, симуляція відбуватиметься тим чи іншим чином.

Як можна бачити, обрану стратегію можна змінювати під час роботи самої програми без необхідності перезапуску:

```
public void initialize(String s) {
    switch(s) {

    case "ComfortZoneStrategy":
        airStrategy = new ComfortZoneStrategy(this.lowerComfortAir, this.upperComfortAir);
        heatingStrategy = new ComfortZoneStrategy(this.lowerComfortTemperature,
                this.upperComfortTemperature);
        break;
    case "HouseIsNotEmptyStrategy":
        for(Room r : manager.getRooms()) {
            r.empty();
            if(Math.round(Math.random()/1.15) == 1) {
                r.addPeople(new RandomPerson(manager.generatePerson(r)));
            }
        }
        heatingStrategy = new HouseIsNotEmptyStrategy();
        lightingStrategy = new HouseIsNotEmptyStrategy();
        break;
    case "PrescribedHoursStrategy":
```

І одразу можна бачити перевагу використання інтерфейсів: конкретні класи ComfortZoneStrategy, HouseISNotEmpty та PrescribedHoursStrategy імплементують по два інтерфейси, адже можуть бути застосовними для різних контролерів.

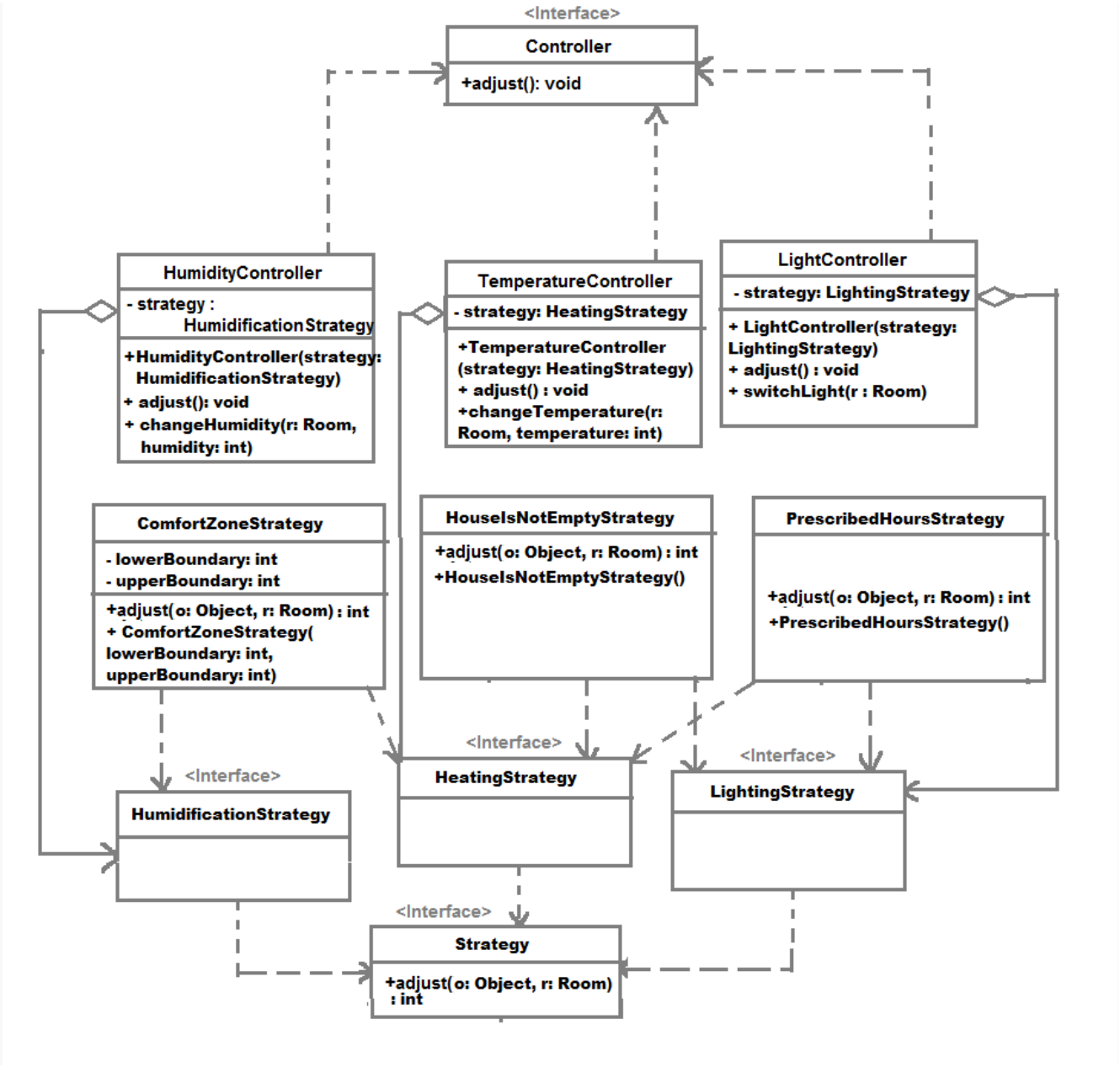

Рис. 8 Реалізація шаблону стратегія

2.2.5 Одинак

Одинак – це породжувальний шаблон проєктування, який гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього[11].

Патерн одинак було використано при створенні класу WindowInitializer, який відповідає за створення форми, на якій користувач малюватиме поверховий план. Також на дану форму будуть нанесені кнопки та випадні списки з інших

частин програми, тому було забезпечено єдиність форми, щоб при виклику ми зверталися до однієї і тої ж канви. А оскільки конструктор класу завжди повертає новий об'єкт, модифікатор доступу до нього було встановлено приватним, таким чином до нього матимуть доступ лише методи усередині класу. Створення об'єкту здійснюється за допомогою статичного методу init, що викликає конструктор при першому створенні об’єкта або повертає попередньо створений об’єкт в іншому випадку.

```
public class WindowInitializer extends JPanel{

    private static WindowInitializer instance;
    private JFrame frame;

    private WindowInitializer(int x, int y) {

    public static WindowInitializer init(int x, int y) {
        if(instance==null) {
            instance = new WindowInitializer(x, y);
        }
        return instance;
    }

    public JFrame getFrame() {

}
```

Це забезпечує глобальну точку доступу до одного об’єкта із усіх частин програми, а також унеможливлює повторну ініціалізацію у випадку, якщо при розширенні функціоналу програми у якійсь ділянці коду буде здійснено спробу створення іще одного об’єкта такого класу.

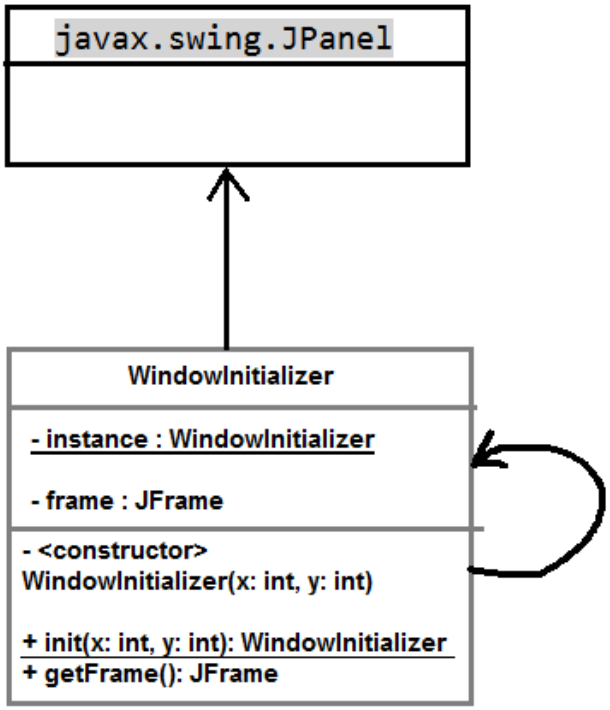

Рис. 9 Реалізація шаблону Одинак

2.2.6 Фасад

Фасад – це структурний шаблон проєктування, який надає простий інтерфейс до складної системи класів, бібліотек або фреймворку[11].

Фасад допомагає відмежувати складну логіку від клієнта. Складність імплементації методів не має впливати на користувачів, тому можна огорнути складну логіку у «обгортку» клас-фасад, а клієнтові дати доступ до методів фасаду, який виконуватиме потрібні операції. Також, щоб розмежувати функціонал за логікою та забезпечити принцип єдиного обов'язку класів, можна ввести додаткові допоміжні фасади, з якими співпрацює основний фасад.

У розробленій програмі користувач у головному класі бачить метод, що малює інтерфейс користувача, але уся логіка того, як відбувається малювання, схована:

```
public static void main(String[] args){
    UIManager uiManager = new UIManager(WindowInitializer.init(1000, 600));
    uiManager.drawUI();
```

Після виклику методу drawUI UIManager делегує роботу із малювання класу-фасаду DrawingFacade, який займається малюванням компонентів та має допоміжні фасади ButtonsManager для відображення кнопок та DropListManager для відображення випадних списків.

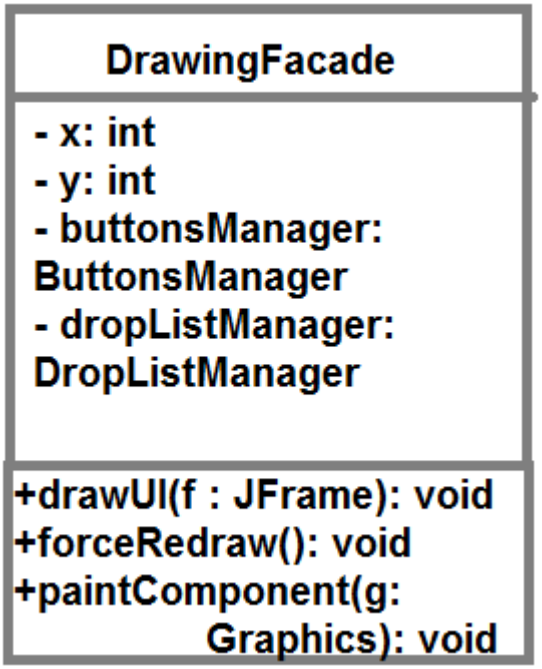

Рис. 10 Реалізація шаблону «Фасад»

Розділ 3.

Демонстрація програми та її основні функції

Для візуальної частини була використана технологія Java Swing та JDK 17. Інтерфейс легкий для сприйняття та користування. Програма є достатньо зрозумілою користувачеві та розроблена таким чином, щоб дозволити моделювати різні випадки.

Оскільки кімнати у будинку можуть мати довільну форму, користувач має змогу самостійно визначити кількість кутів та межі. Для зручнішого відображення площі, канву розграфлено із поділом на умовні квадратні метри. Таким чином можна реалістично зобразити приміщення у масштабі.

У випадку, коли необхідно перемалювати план, реалізована кнопка «Clear canvas», яка очищує полотно, і тоді можна малювати заново без необхідності перезапускати програму.

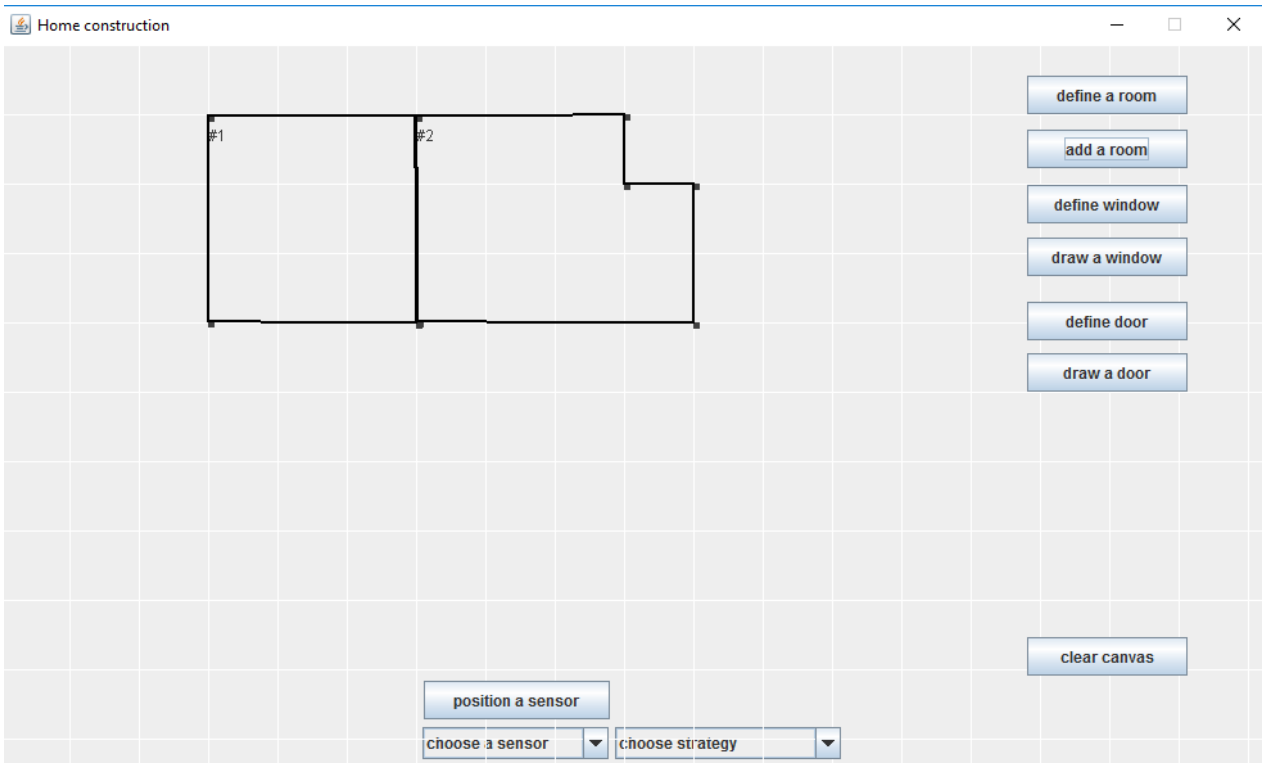

Рис. 11 Додавання кімнат довільної форми на канву.

Програма дозволяє спроєктувати необхідний поверховий план та розмістити вікна та двері: для цього було реалізовано блок кнопок справа із прив’язаними відповідними слухачами подій. Також є випадний список із сенсорами, за допомогою якого можна обрати тип сенсора, а також розмістити його на плані. На нижній частині панелі є випадний список, що дозволяє обрати один із сценаріїв роботи.

Немає жодних обмежень щодо форми кімнати, проте, коли кімнату визначено, вікна та двері можна розташувати лише на визначених стінах. Програма перевіряє відстань до стіни(має бути не більша за похибку) і не дозволяє розміщення конструктивних елементів усередині приміщення.

```
@Override
public boolean isOnTheWall(Point p) {
    Line line = new Line(this.getFirstPoint(), this.getSecondPoint());
    double wallLength = line.segmentLength();
    Line line1 = new Line(p, this.getFirstPoint());
    double length1 = line1.segmentLength();
    Line line2 = new Line(p, this.getSecondPoint());
    double length2 = line2.segmentLength();
    return(Math.abs(wallLength - (length1 + length2)) < E);
}
```

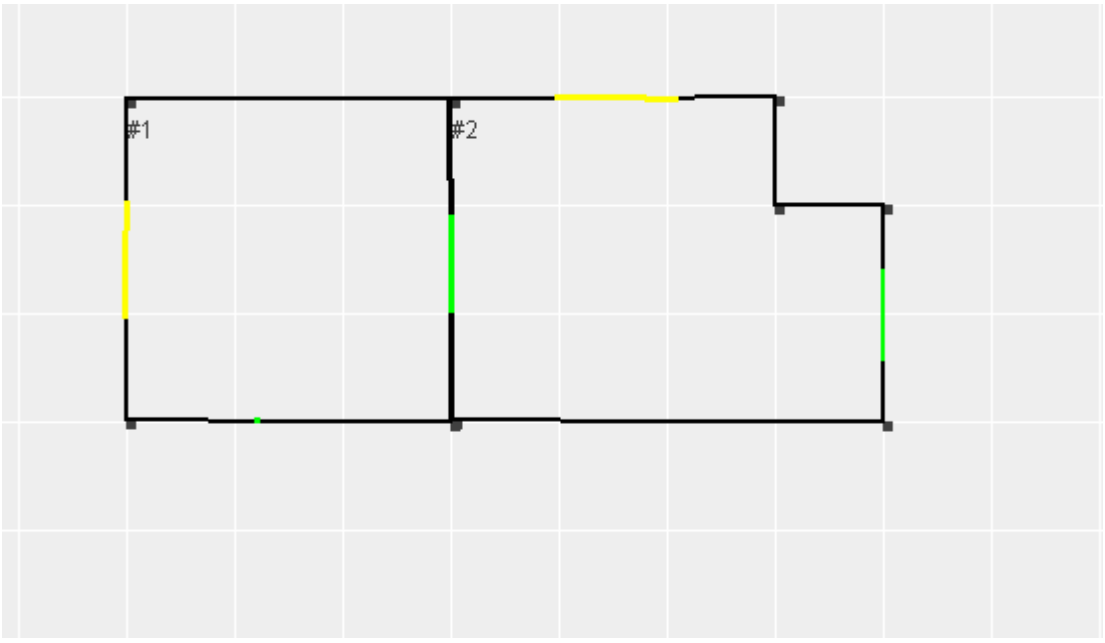

Рис.12 Додавання дверей(зелений) та вікон(жовтий) до поверхового плану

3.1 Сенсори

Сенсори відповідають за сприйняття інформації з навколишнього середовища, яку потім опрацюють контролери. Існує широкий діапазон різних сенсорів: для вимірювання температури, вологості, світла, руху, шуму, витоку газу та ін.

У програмі було розроблено чотири типи сенсорів: ті, що вимірюють температуру, вологість та наявність світла, а також детектори руху. Останні вимірюють величину теплового сліду. Програма має два варіанти обробки отриманих даних: сповіщення користувача про будь-яку діяльність у якійсь із кімнат або сповіщення користувача у випадку, якщо тепловий слід більший за похибку на домашнього улюбленця.

При ініціалізації кожній із кімнат випадковим чином присвоюється температура та вологість із діапазону, а також світло. За замовчуванням усі кімнати порожні. Вивід інформації про кімнату здійснюється у консоль, також можливий вивід у файл, візуальна репрезентація – на формі.

```
Room#1 22*C 66% humidity light off room is empty
Room#2 26*C 62% humidity light on room is empty
Room#3 25*C 35% humidity light off room is empty
Room#4 26*C 62% humidity light on room is empty
```

Програма дозволяє розташувати сенсори у кімнатах, враховуючи їх особливості - сенсори тепла не варто розташовувати на протязі, біля вікон, бо це може привести до неправильного визначення температури кімнати[32].

Тому у програмі присутня перевірка на розташування сенсора, яка впливатиме на точність показників.

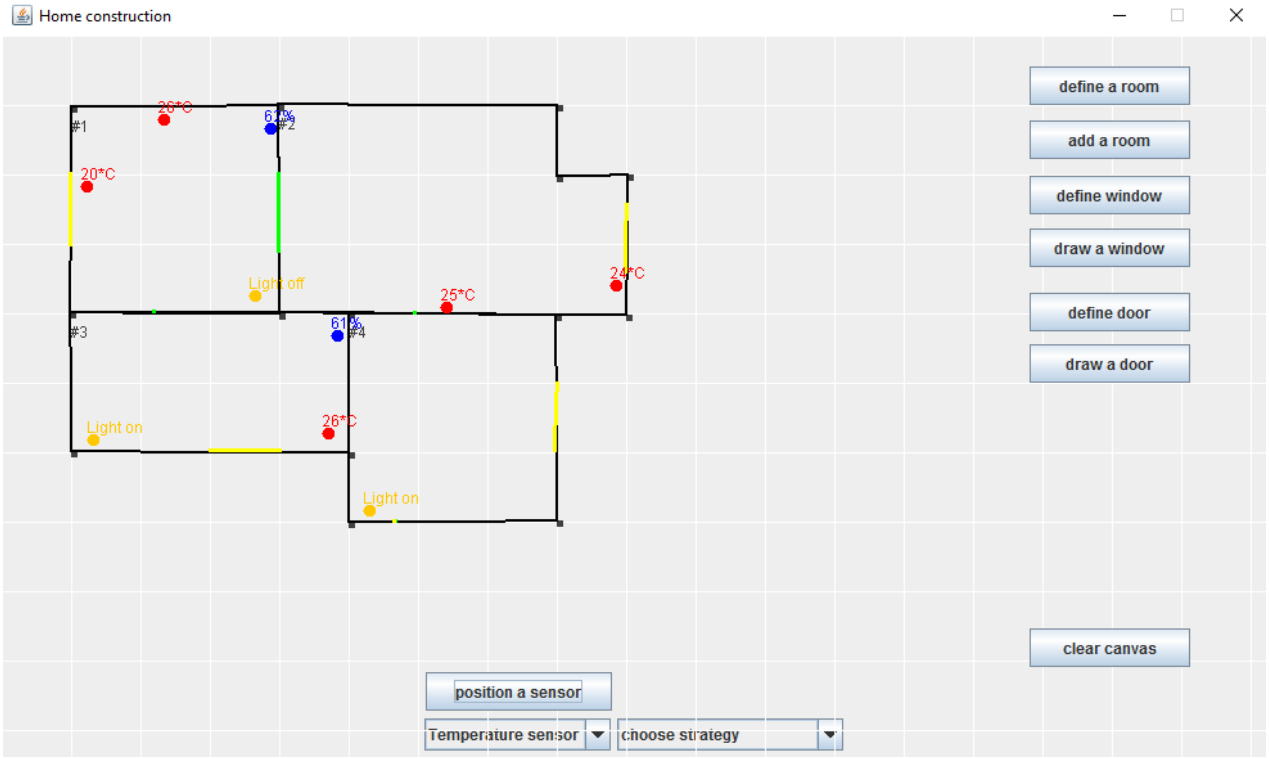

Рис 13 Розміщення різноманітних датчиків

Як видно на малюнку, у кімнаті номер 1 розташовані два сенсори температури: один біля вікна, інший – ні. Той, що знаходиться біля вікна, спотворює значення поточної температури, з чого користувачеві видно, що такі місця для розташування сенсорів несприятливі.

Сенсори можна лише усередині кімнат, а не будь-де на канві: при спробі розташувати сенсор, програма перевіряє, чи знаходиться відповідна позиція у многокутнику, утвореному стінами однієї з кімнат за допомогою процесу, описаного у [31]. Якщо місце, обране користувачем, знаходиться поза межами окреслених кімнат, то сенсор не створюється.

В [1] було реалізовано перевірку на приналежність точки опуклому многокутнику, але оскільки кімната може мати довільну форму, у цій праці алгоритм визначення було модифіковано і було реалізовано алгоритм локалізації точок для довільних многокутників

.

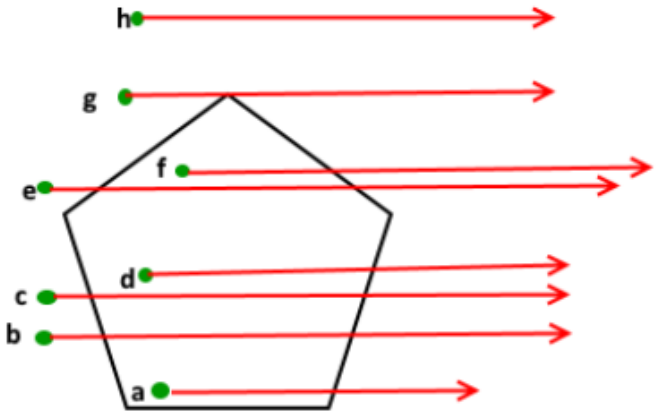

Рис. 14 Перевірка чи належить точка опуклому многокутнику.

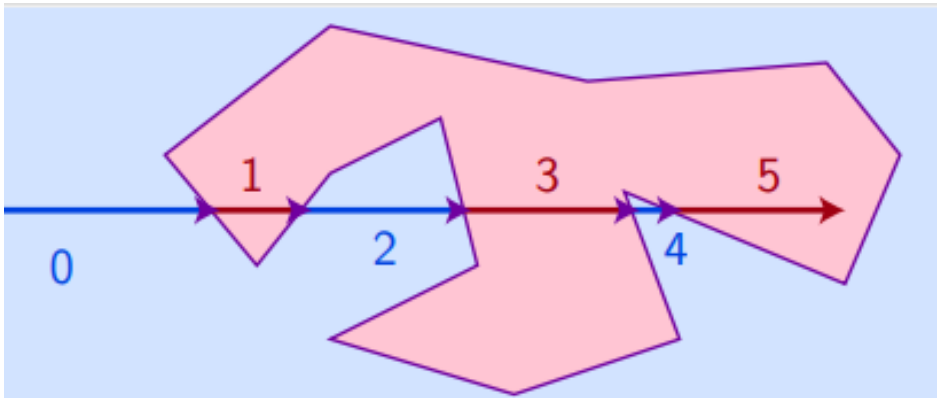

Рис. 15 Перевірка приналежності точки довільному многокутнику[31]

Через точку проводять горизонтальний промінь і обчислюється кількість точок перетину утвореного променя зі сторонами многокутника(тобто з відрізками, які задають стіни помешкання). Якщо отримане число парне, то точка знаходиться поза межами многокутника. Якщо кількість перетинів непарна, то точка належить внутрішній області многокутника[31], тож сенсор можна розташовувати.

```
@Override
public boolean isInsideTheRoom(Point p) {
    int count = 0;
    Point p2 = new Point(WindowInitializer.init(0, 0).getFrame().getWidth(),
            (int) p.getY());
    for(Point c: corners) {
        if(Math.abs(c.getX() - p.getX()) + Math.abs(c.getY() - p.getY()) < E) {
            return false;
        }
    }
    Line l1 = new Line(p, p2);
    for(Wall w : walls) {
        if(w.isOnTheWall(p)) {
            return false;
        }
        Line l2 = new Line(w.getFirstPoint(), w.getSecondPoint());
        if(l1.segmentIntersect(l2)) {
            count++;
        }
    }
    return( count%2!=0);
}
```

Рис. 16 Перевірка приналежності точки внутрішній області довільного многокутника

3.2 Контролери

Контролери отримують інформацію із розміщених сенсорів та опрацьовують інформацію відповідно до заданого алгоритму дій. У реалізованій програмі вони керують процесом відпрацювання сценаріїв. Було створено чотири класи контролерів : для управління світлом, опаленням та вологістю приміщення, керування переміщеннями. Контролери зчитують дані із сенсорів та виконують передбачені дії(вмикають/вимикають опалення, світло, зволожувач повітря) допоки показники не досягнуть визначеної норми. Норма визначається одним із сценаріїв, що обирає користувач.

Стратегія «ComfortZoneStrategy» слідкує за тим, щоб температура чи вологість були в межах заданих комфортних меж та корегує дані, якщо вони занадто низькі чи високі.

Стратегія «HouseIsNotEmptyStrategy» корегує дані, залежно від того, чи є хтось у приміщенні – пусту кімнату не має сенсу опалювати на повну потужність чи освітлювати, таким чином здійснюється економія енергоресурсів.

Стратегія «PrescribedHoursStrategy» вмикає або вимикає опалення та/чи освітлення з настанням певної години.

Наприклад, за умови обрання стратегії «Зона комфорту» контролери керуватимуть мікрокліматом кімнати у межах показників комфортності – межі комфортної температури та вологості. Були встановлені межі комфортності вологості від 30 до 45 відсотків, температури – від 22 до 25 градусів.

За початкових даних:

```
Room#1 20*C 33% humidity light on room is empty
Room#2 23*C 32% humidity light off room is empty
Room#3 19*C 63% humidity light off room is empty
Room#4 19*C 33% humidity light on room is empty
```

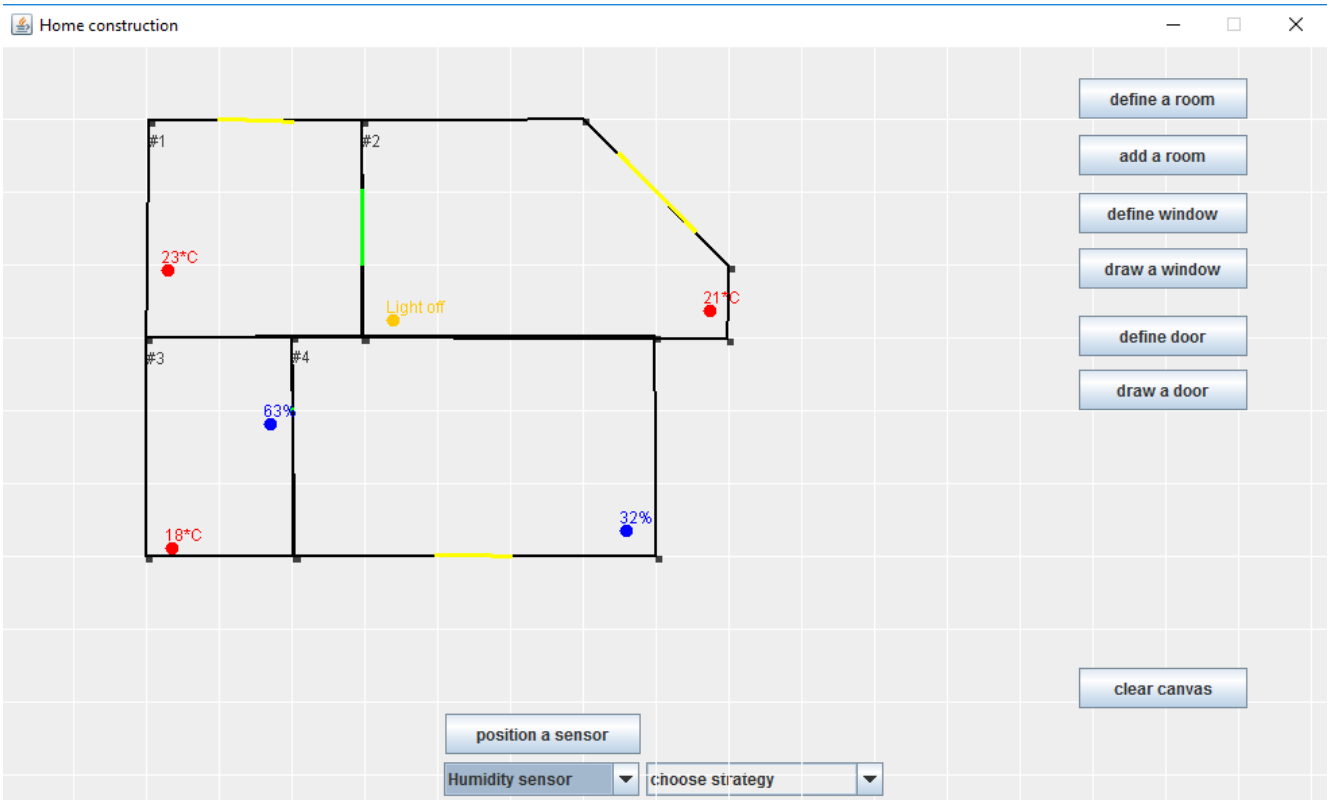

Рис. 17 Початкові дані перед опрацюванням сценаріїв

Було отримано такі показники після застосування сценарію «Зона Комфорту»:

```
Room#1 23*C 33% humidity light on room is empty
Room#2 22*C 32% humidity light off room is empty
Room#3 22*C 45% humidity light off room is empty
Room#4 19*C 32% humidity light on room is empty
```

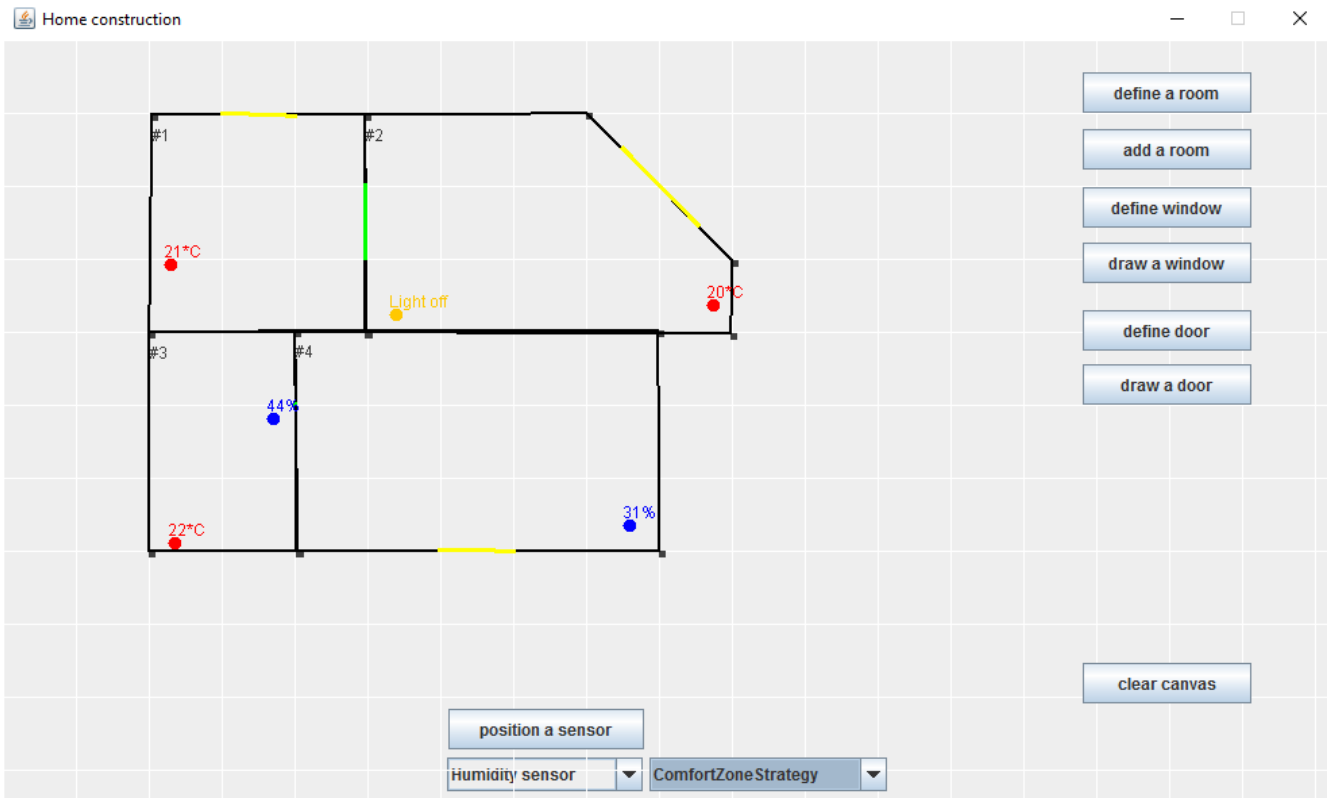

Рис. 18 Після відпрацювання стратегії «Зона комфорту»

Усі показники знаходяться в межах комфортних умов, проте у кімнаті номер чотири не було розміщено датчик температури, тому контролер не отримав інформації про температуру у кімнаті, і її не було відрегульовано відповідно до бажаної. Таким чином користувач бачить, що необхідно розмістити сенсори тепла в усіх кімнатах, де він часто перебуває, а отже в яких йому важлива температура саме з заданого діапазону.

При застосуванні сценарію «Хтось є вдома», програма випадковим чином поміщає людей у кімнати і тоді контролери залежно від цих даних регулюють температуру та світло.

```
Room#1 18*C 33% humidity light on room is empty
Room#2 22*C 32% humidity light on room is not empty
Room#3 19*C 45% humidity light off room is empty
Room#4 19*C 32% humidity light on room is not empty
```

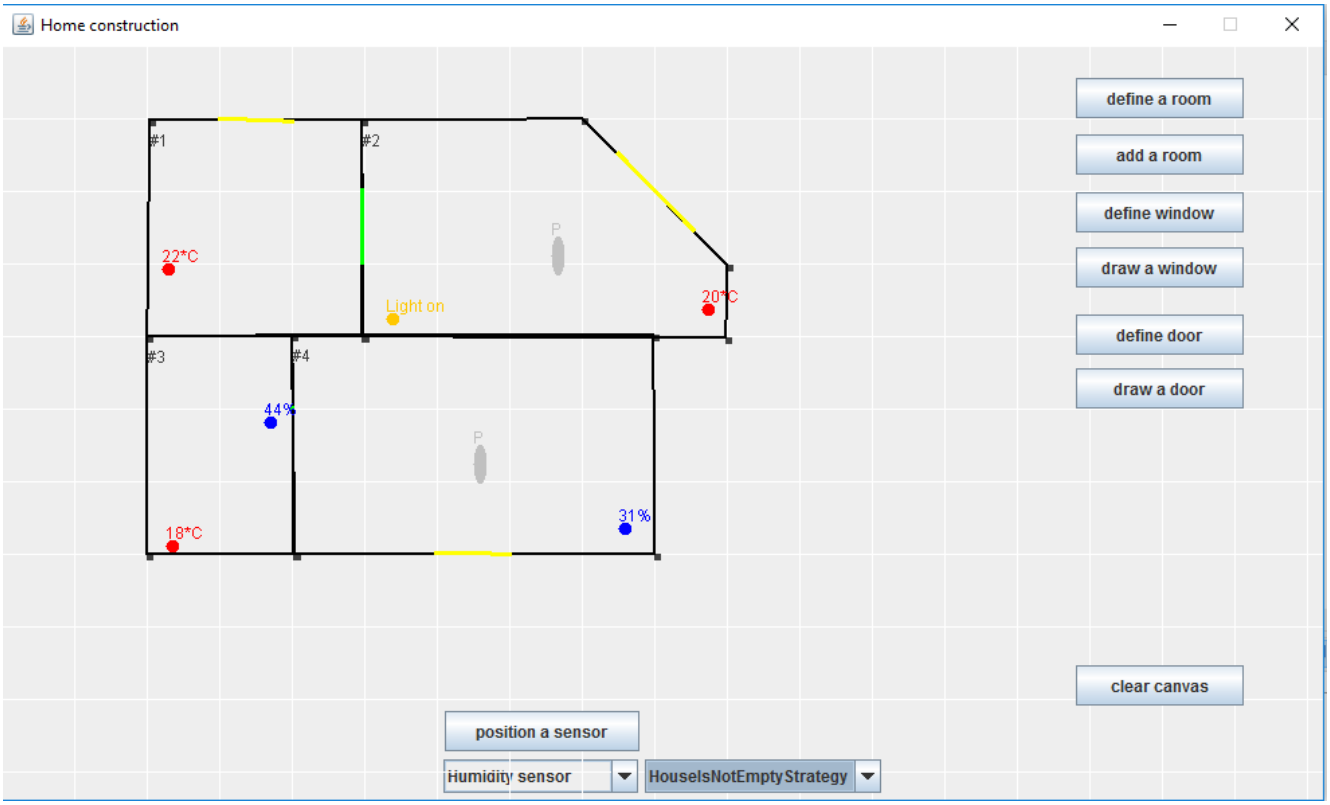

Рис. 19 Після відпрацювання стратегії «Хтось є вдома»

Як бачимо, у тих приміщеннях, де випадковим чином були згенеровані мешканці, було підвищено температуру, у тих, де нікого немає – понижено. Важко проаналізувати реакцію світлових контролерів, бо датчики світла були не у всіх кімнатах.

Тому додамо датчики світла в усі кімнати для наочності:

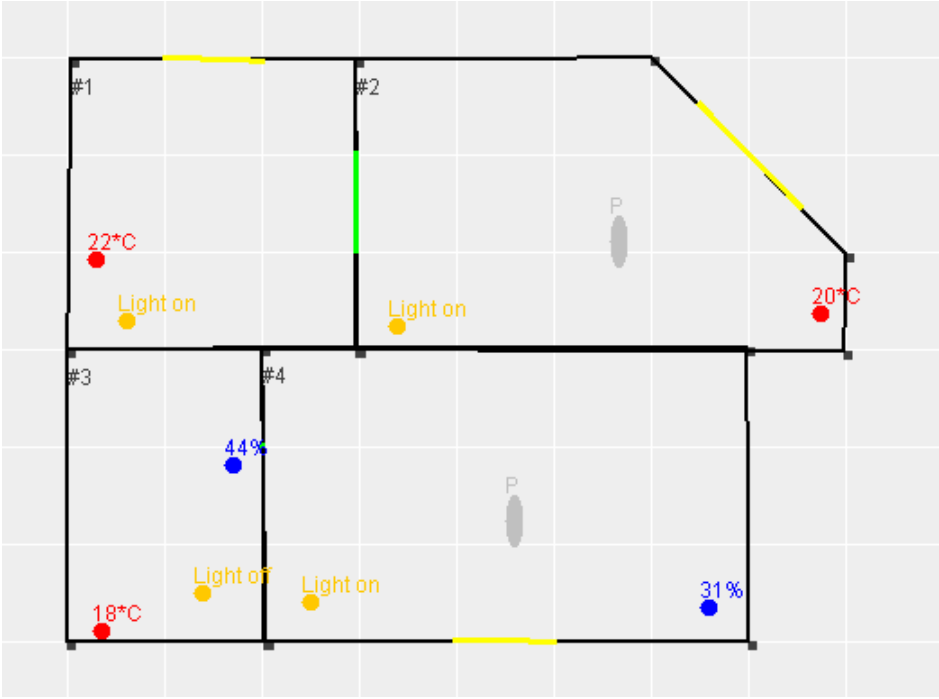

Рис. 20 Початковий стан після розміщення датчиків світла

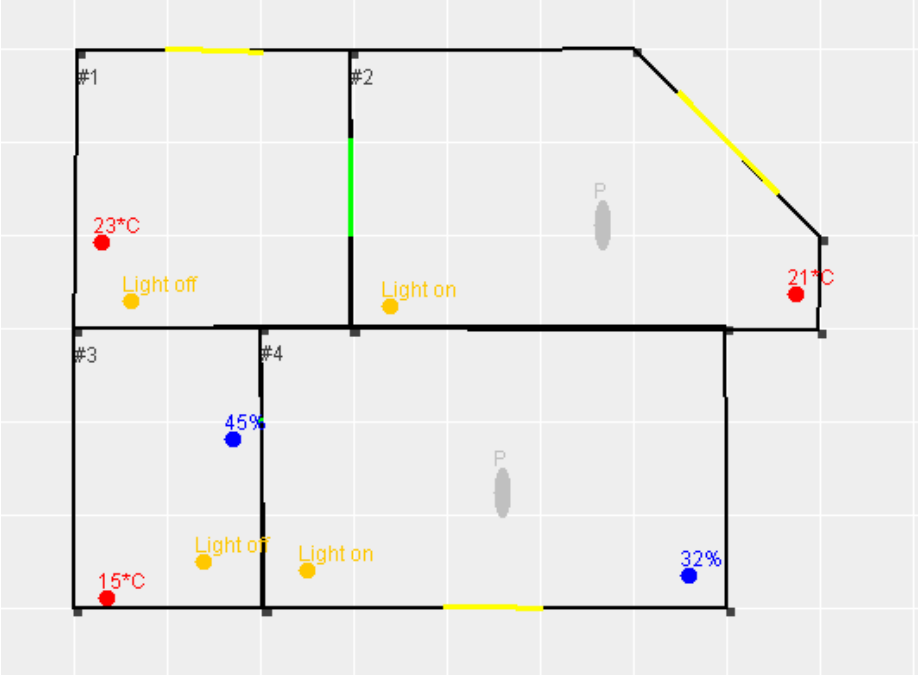

Рис. 21 Стан після відпрацювання сценарію «Хтось є вдома»

Після застосування стратегії «Хтось є вдома» добре видно, що в усіх порожніх кімнатах вимкнено світло та зменшена температура.

Стратегії також можна комбінувати і застосовувати одна після одної: наприклад відрегулювати показники до комфортного рівня із «Зоною комфорту», а потім підтримувати енергозбереження за допомогою стратегії «Хтось є вдома».

Також було протестовано датчики руху. Програма випадковим чином генерує відсутність чи наявність живої істоти в приміщенні. Датчик руху реагує на великий об’єкт(людину), але не реагує на маленький об’єкт(домашній улюбленець)

```
Room#1 24*C 73% humidity light off room is empty
Room#2 15*C 66% humidity light off room is empty
Room#3 23*C 62% humidity light off room is not empty
```

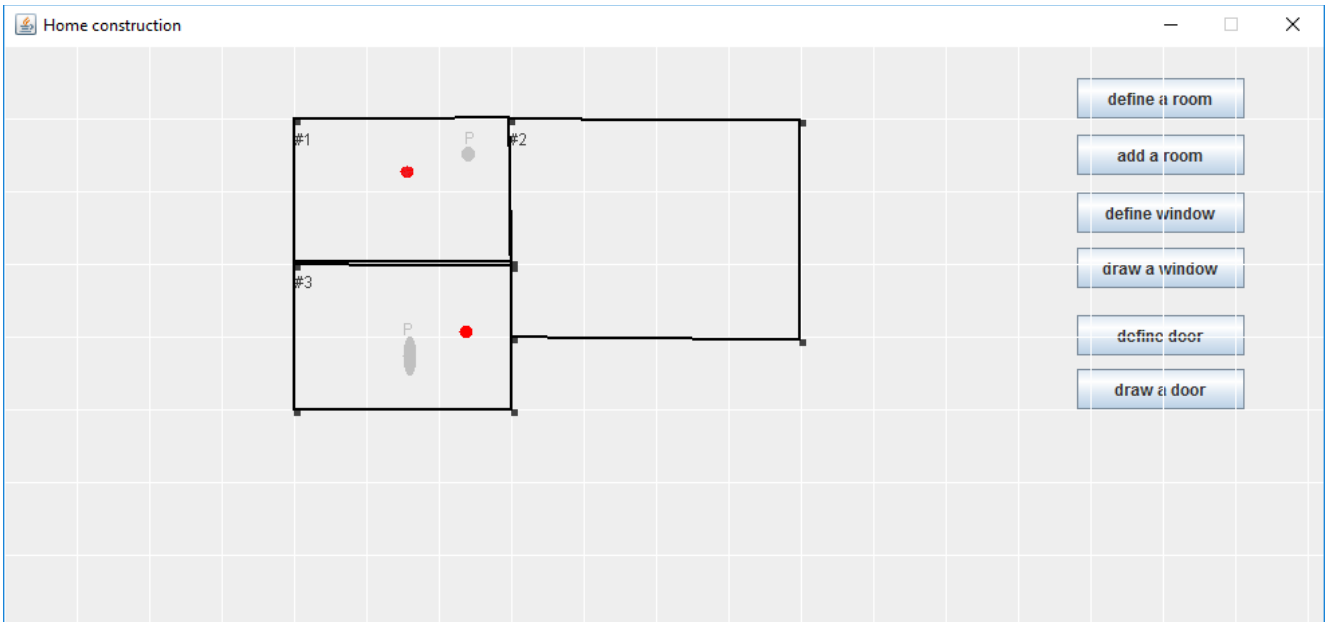

Рис. 22 Відпрацювання шаблону спостерігач, що відповідає за безпеку у даній програмі

Висновки

Шаблони проектування – це важливий інструмент для покращення якості коду, який сприяє його повторному використанню, а також полегшує розширення функціоналу програми.

Розвиток сучасних технологій призводить до все більшої автоматизації побуту, тому система «розумний будинок» з часом може стати частиною буденного життя пересічних людей, дозволяти їм економити грошові та енергетичні ресурси.

За допомогою розробленої програми можна протестувати роботу такої системи та підібрати датчики під свої потреби. Користувач може спроектувати лаконічний поверховий план наявної чи планованої нерухомості.

Програму із [1] було розширено із додаванням нового функціоналу, що показує ефективність шаблонів проєктування.

Список використаної літератури

1. «Моделювання системи «розумний дім» з використанням шаблонів проєктування», курсова робота / Крохіна Єлизавета // - травень 2022 – 25 с.
2. https://ncube.com/blog/current-smart-home-technologies
3. https://www.tantiv4.com/insights/iot-big-stories/5-must-have-features-of-a-smart-home
4. https://www.epravda.com.ua/publications/2020/07/20/663103/
5. https://pro-consulting.ua/ua/pressroom/um-kotoryj-mozhno-kupit-za-dengi-analiz-rynka-sistem-umnyj-dom-v-ukraine
6. https://pro-consulting.ua/ua/pressroom/rynok-umnogo-doma-v-ukraine-komfort-i-bezopasnost-stoit-svoih-deneg
7. https://www.reuters.com/technology/tesla-workers-shared-sensitive-images-recorded-by-customer-cars-2023-04-06/
8. https://www.bbc.com/ukrainian/features-64008306
9. «Моделювання системи «розумний дім» з використанням шаблонів проєктування», курсова робота / Крохіна Єлизавета // - листопад 2022 – 17 с.
10. https://www.codingame.com/playgrounds/503/design-patterns/origin-of-design-patterns
11. Занурення в патерни проектування / Олександр Швець // - 2021 – 396 с.
12. https://www.afcdud.com/fr/smart-city/422-how-the-history-of-smart-homes.html
13. https://smarthomeenergy.co.uk/the-history-of-the-smart-home/
14. https://computerhistory.org/blog/the-echo-iv-home-computer-50-years-later/
15. https://retailtechinnovationhub.com/home/2022/10/11/four-reasons-why-smart-home-tech-retail-is-booming
16. https://www.vivint.com/resources/article/home-tips-for-energy-efficiency
17. https://earthweb.com/smart-home-statistics/
18. https://www.statista.com/statistics/1056057/worldwide-smart-home-security-market-value/

19. https://www.fortunebusinessinsights.com/industry-reports/smart-home-market-101900
20. https://www.javatpoint.com/structural-design-patterns
21. https://www.sealights.io/software-quality/software-maintainability-what-it-means-to-build-maintainable-software/
22. https://www.tutorialspoint.com/software_quality_management/software_quality_management_metrics.htm
23. https://www.ibm.com/docs/en/raa/6.1?topic=metrics-cyclomatic-complexity
24. https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-maintainability-index-range-and-meaning?view=vs-2022
25. https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-class-coupling?view=vs-2022
26. https://refactoring.guru/uk/design-patterns/history
27. https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2022
28. Evaluating Impact of Design Patterns on Software Maintainability and Performance / Farooq Abdullah// - 2017 – 104 c.
29. Smart home systems, Branko Dvoršak
30. https://www.semanticscholar.org/paper/Energy-conservation-in-a-smart-home-Tejani-Al-Kuwari/7e2b559e3f4c3f49469a38b1eb5056834f036066
31. http://geometry.karazin.ua/resources/documents/20150427134002_227176b96dd42.pdf
32. https://ds-electronics.com.ua/ua/support/blog/termoregulyatory/gde-ustanovit-termoregulator-i-datchik-tepla-v-zavisimosti-ot-prednaznachenija/
33.