Міністерство освіти і науки України

Національний авіаційний університет

Факультет кібербезпеки, комп’ютерної та програмної інженерії Кафедра комп’ютерних інформаційних технологій

Курсовий проект

З дисципліни: «Об’єктно-орієнтоване програмування»

На тему: «Засоби контролю та управління паливною системою літака АН-140»

Виконали студенти групи ТП-216Б:

Мартич Дмитро, Василенко Єлизавета,

Ляпін Ангеліна, Павло Кужель,

Киян Ростислав

Прийняв:

Єрмачков Ю.О.

Київ 2021

**Зміст**

[1. Постановка завдання 2](#_Toc90835372)

[2. Варіант виконання курсової роботи 3](#_Toc90835373)

[3. Уривок керівництва з льотної експлуатації АН-140 3](#_Toc90835374)

[4. Діаграми UML об’єктної моделі системи 5](#_Toc90835375)

[4.1. Діаграма варіантів використання 5](#_Toc90835376)

[4.2. Діаграма послідовності 7](#_Toc90835377)

[4.3. Діаграма кооперацій 10](#_Toc90835378)

[4.4. Діаграми класів 13](#_Toc90835379)

[4.5. Діаграми станів 16](#_Toc90835380)

[5. Програмні засоби 18](#_Toc90835381)

[6 Програмна реалізація 20](#_Toc90835382)

7.Результат виконання програми ............................................................... 23

8.Висновок ................................................................................................... 30

9. Список літератури .....................................................................................31

# Постановка завдання

Розробити об’єктну модель системи «Засоби контролю та управління паливною системою літака Ан-140» **(**реалізувати взаємодію оператора з паливною системою літака Ан-140 на різних етапах її експлуатації згідно «Руководству по летной эксплуатации самолета Ан-140», частина 8.3 (Топливная система), розділ «Нормальная эксплуатация») звикористанням мови візуального моделювання UML та її програмну реалізацію із застосуванням мови програмування Java згідно з варіантом завдання.

Об’єктна модель системи повинна включати в себе потрібні діаграми UML з нижче вказаного переліку:

* варіантів використання (Use Case Diagram);
* дій (Activity Diagram); (послідовностей та кооперацій)
* класів (Class Diagram); (робити після діаграми послідовностей)
* станів (Statechart Diagram);
* послідовності (Sequence Diagram);
* кооперацій (Collaboration Diagram);
* компонентів (Component Diagram);
* розгортання (Deploument Diagram).

# Варіант виконання курсової роботи

Варіант №2

|  |  |
| --- | --- |
| **Зміст завдання (назва етапу експлуатації)** | **Дії оператора** |
| «Включение подачи топлива к двигателям перед запуском двигателей» (Додаток 2) | Откройте пожарные краны двигателей, установив на щитке «ПОЖАРНЫЕ КРАНЫ» два переключателя в верхнее положение, Кратковременно загорятся и погаснут светосигнализаторы «НЕ ЗАКРЫТ», загорятся светосигнализаторы «ОТКРЫТ».  Зафиксируйте переключатели колпачками  Нажмите четыре кнопки-табло «№1 НАСОС №2» – включатся насосы, загорятся зеленые поля кнопок-табло    Погаснут табло «ДВИГ 1 (2) – ДАВЛ  ПОДКАЧ НЕТ» |

# Уривок керівництва з льотної експлуатації АН-140

**Ан-140**— регіональний вантажо-пасажирський авіалайнер[,](https://uk.wikipedia.org/wiki/%D0%90%D0%B2%D1%96%D0%B0%D0%BB%D0%B0%D0%B9%D0%BD%D0%B5%D1%80) розроблений АНТК ім. Антонова. Призначений для пасажирських і змішаних вантажопасажирських перевезень на відстань до 3 700 кілометрів. Здійснив перший політ 17 вересня 1997. Літак є високопланом [з](https://uk.wikipedia.org/wiki/%D0%92%D0%B8%D1%81%D0%BE%D0%BA%D0%BE%D0%BF%D0%BB%D0%B0%D0%BD) класичним оперенням і двоматурбогвинтовими двигунами ТВ3-117ВМА-СБМ1[,](https://uk.wikipedia.org/wiki/%D0%A2%D0%923-117%D0%92%D0%9C%D0%90-%D0%A1%D0%91%D0%9C1) встановленими накрилі.Допоміжна силова установка АІ9-3Б, розміщена в хвостовій частиніфюзеляжу, забезпечує автономну експлуатацію літака на необладнанихаеродромах. У хвостовій частині фюзеляжу і під підлогою пасажирської кабіни розміщенібагажноватажні [в](https://uk.wikipedia.org/wiki/%D0%92%D0%B0%D0%BD%D1%82%D0%B0%D0%B6)ідсіки загальним об'ємом 9,1м³ і [вантажопідйомністю 1](https://uk.wikipedia.org/wiki/%D0%92%D0%B0%D0%BD%D1%82%D0%B0%D0%B6%D0%BE%D0%BF%D1%96%D0%B4%D0%B9%D0%BE%D0%BC%D0%BD%D1%96%D1%81%D1%82%D1%8C),84 тонни. Об'єми багажно-вантажних відсіків у 1,3-1,5 рази більше, ніж в інших літаках аналогічного класу.

Паливна система літака призначена для розміщення палива і здачі його до двигунів і ЗСУ на всіх режимах експлуатації літака.

Паливна система включає в себе:

* Паливні баки;
* Систему дренажу баків;
* Систему централізованої заправки;
* Систему вироблення палива; • Паливо-вимірювальну систему (ТІС);
* Органи управління та контролю.

Електроживлення паливної системи проводиться від наземних і бортових джерел електроживлення постійного струму напругою 27В. Паливо літаком розміщається у двох крилах баках-кесонах, по одному в кожному напівкрилі. Кожен бак нервюрами №13 і 15 розділений на три відсіки: «предрасходный», насосний і консольна частина витратного відсіку. Насосний відсік разом з консольної частиною витратного відсіку складає видатковий відсік. Кожен двигун живиться паливом з бака свого напівкрила. У магістралях подачі палива до двигунів встановлені пожежні крани.

Перед пожежними кранами паливні трубопроводи обох двигунів з'єднані між собою магістраллю кільцювання з краном кільцювання. Це дозволяє, при необхідності, подавати паливо з бака одного двигуна для живлення іншого (або 2-х двигунів з одного бака) при відкритому крані кільцювання.

У нижніх частинах нервюр №13 і 15 встановлені по три зворотних межбакового клапана, дозволяючи перетікання палива в насосний відсік з двох суміжних з ним.

Дренаж кожного бака здійснюється своєї автономною системою дренажу. Повідомлення відсіків бака з атмосферою забезпечується дренажним відсіком, розташованим в окремій частині крила.

У дренажному відсіку встановлений повітрязабірник. Для оберігання бака від руйнування при закритому повітрязабірнику дренажний відсік повідомляється з атмосферою через вакуумний клапан і клапан надлишкового тиску. Трубопроводом дренажний відсік з'єднаний з «предрасходным» відсіком і забезпечує його дренаж в будь-якому положенні літака.

Дренаж насосного відсіку здійснюється через отвір у верхній частині нервюри №15, а - також через дренажний трубопровід з жиклером.

Дренаж забезпечується також меж бакового зворотними клапанами, встановленими в нижній частині нервюр № 13 і 15. Паливо, що потрапило при еволюціях літака в дренажний відсік, зливається по трубопроводу в «предрасходный» відсік.

Паливо-вимірювальна система встановлена на літаку, забезпечує виконання таких функцій:

* Постійне вимірювання і обчислення маси палива в кожному баку;
* Автоматичне закриття кранів заправки і заслінки при заповненні баків паливом;
* Видачу інформації про масі палива в баках, про сумарну масу палива на літаку на цифрові індикатори пультів управління виробленням і заправкою, аварійний реєстратор і літаковий відповідач;
* Видачу сигналу резервного залишку палива в систему аварійних сигналізаторів і аварійний реєстратор; • Формування сигналу "ДИСБАЛАНС ПАЛИВ";
* Видачу сигналу про досягнення встановленої температури палива;
* Видачу інформації про температуру палива;
* Видачу інформації про несправності в каналах паливовимірювання, вимірювання температури та інформації про відмови насосів;
* Ручне управління краном кільцювання і електропривідними насосами.

# Діаграми UML об’єктної моделі системи

## Діаграма варіантів використання

Одна з моделей формалізації процесу постановки цілей і завдань проекту була запропонована фірмою Rational і ввійшла в стандарт мови UML. Це діаграми варіантів використання (use case), які іноді називають діаграмами прецедентів. Варіант використання являє собою типову взаємодію користувача й проектованої системи і характеризується такими властивостями:

* варіант охоплює деяку очевидну для користувачів функцію;
* варіант може бути як невеликим, так і досить великим;
* варіант вирішує деяке дискретне завдання користувача.

У найпростішому випадку варіант використання створюється в процесі обговорення з користувачами тих речей, які вони хотіли б одержати від системи [5]. При цьому кожній окремій функції привласнюється ім'я й записується її короткий текстовий опис.

Це все, що необхідно у фазі аналізу. Знання деяких деталей може знадобитися, якщо передбачається, що даний варіант використання містить важливі архітектурні відгалуження. Більшість варіантів використання може бути деталізована під час конкретної ітерації в процесі проектування.

Основними елементами діаграми варіантів використання є діючі особи, варіанти використання й відносини між ними. Діюча особа - це роль, що користувач грає відносно системи.

Під час аналізу і проектування варіанти використання дозволяють зрозуміти як результати, які хоче отримати користувач впливають на архітектуру системи і як повинні поводитися компоненти системи, для того, щоб реалізувати потрібну для користувача функціональність.

Варіант використання являє собою типову взаємодію користувача й проектованої системи і характеризується такими властивостями:

* варіант охоплює деяку очевидну для користувачів функцію;
* варіант може бути як невеликим, так і досить великим;
* варіант вирішує деяке дискретне завдання користувача.

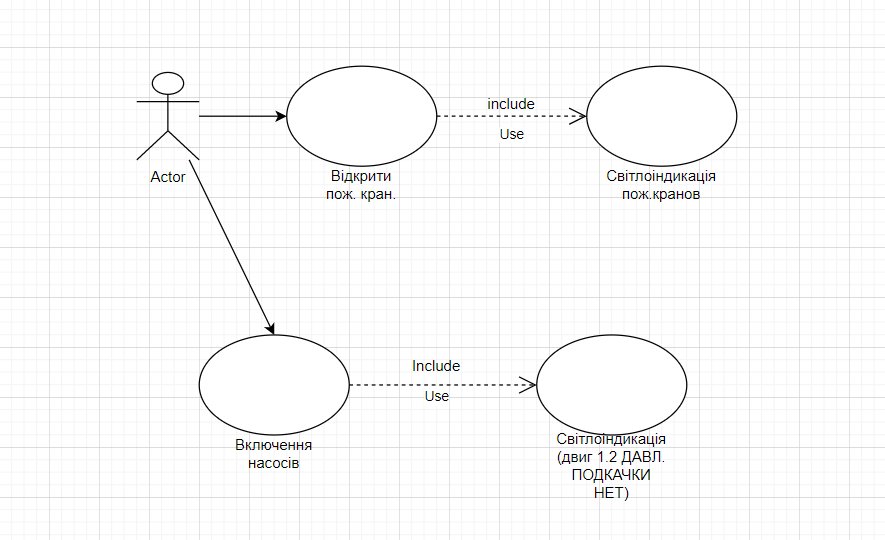
В процесі тестування, описані раніше варіанти використання дозволяють простіше оцінити точність реалізації вимог користувачів і дозволяють провести покрокову перевірку цих вимог.

Стратегія використання прецедентів при визначенні вимог визначає необхідність додатково до питання "що користувачі чекають від системи?" ставити питання "що система повинна зробити для конкретного користувача?".

Такий підхід дозволяє шукати функції, які потрібні багатьом користувачам і виключати ті можливості, які не можуть допомогти користувачам виконувати свої повсякденні завдання.

*Склад діаграми Use Case:*

Діаграма варіантів використання складається з акторів, для яких система проводить дію і власне дії Use Case, яка описує те, що актор хоче отримати від системи. Актор позначається значком чоловічка, а Use Case - овалом. Додатково в діаграми можуть бути додані коментарі.



*Рис.1. Use-case діаграма системи «Засоби контролю та управління паливною системою літака Ан-140»*

## Діаграма послідовності

На діаграмі послідовності зображаються виключно ті об'єкти, які безпосередньо беруть участь у взаємодії і не показуються можливі статичні асоціації з іншими об'єктами. Для діаграми послідовності ключовим моментом є саме динаміка взаємодії об'єктів в часі. При цьому діаграма послідовності має як би два виміри. Одне - зліва направо у вигляді вертикальних ліній, кожна з яких зображує лінію життя окремого об'єкта, який бере участь у взаємодії. Графічно кожен об'єкт зображується прямокутником і розташовується у верхній частині своєї лінії життя. Усередині прямокутника записуються ім'я об'єкту і ім'я класу, розділені двокрапкою. При цьому вся запис підкреслюється, що є ознакою об'єкта, який, як відомо, являє собою екземпляр класу. Кожен об'єкт існує тільки 1 раз від моменту створення до знищення. Деякі об'єкти вважаються вже створеними (показуються вгорі).

Види повідомлень:

* Синхронні або складові повідомлення (типу виклику процедури) може бути складовим (з влож. Викликами)
* Просте повідомлення (сигнал)
* Повідомлення про завершення деякого дії чи процедури повернення будь-якого значення << Return >> стрілка повернення може не малюватися, якщо вона очевидна, щоб не ускладнювати діаграму Розрізняють активні і пасивні об'єкти. *активний* - має свою нитку управління. вводиться *фокус управління*. Об'єкти можуть самознищуватися або знищуватися іншими об'єктами.

*Повідомлення* - фундаментальне поняття UML. Повідомленнями обмінюються лінії життя об'єктів. У послідовності обміну повідомленнями виникає *траєкторія подій*. Діаграма послідовності будується по 1 основний траєкторії без альтернатив

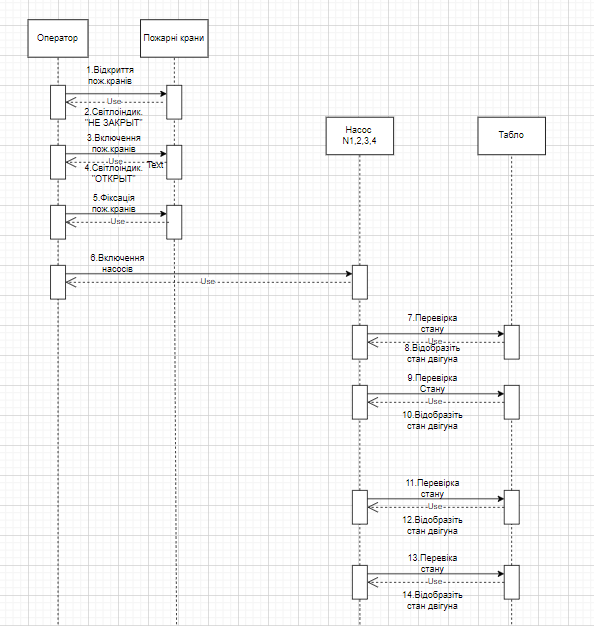
Лінія життя об'єкту (object lifeline) зображується пунктирною вертикальною лінією, асоційованою з єдиним об'єктом на діаграмі послідовності. Лінія життя служить для позначення періоду часу, протягом якого об'єкт існує в системі і, отже, може потенційно брати участь у всіх її взаємодіях. Якщо об'єкт існує в системі постійно, то і його лінія життя повинна тривати по всій площині діаграми послідовності від самої верхньої її частини до найнижчої. Зовсім не обов'язково створювати всі об'єкти в початковий момент часу.

Окремі об'єкти в системі можуть створюватися у міру необхідності, істотно економлячи ресурси системи і підвищуючи її продуктивність. В цьому випадку прямокутник такого об'єкта зображується не в верхній частині діаграми послідовності, а в тій її частині, яка відповідає моменту створення об'єкта (об'єкт 6 на рис.). При цьому прямокутник об'єкта вертикально розташовується в тому місці діаграми, яке по осі часу збігається з моментом його виникнення в системі. Очевидно, об'єкт обов'язково створюється зі своєю лінією життя і, можливо, з фокусом управління. Як вже зазначалося, побудова діаграми послідовності доцільно починати з виділення з усієї сукупності тих і тільки тих класів, об'єкти яких беруть участь в моделируемом взаємодії. Після цього всі об'єкти наносяться на діаграму з дотриманням деякого порядку ініціалізації повідомлень. Тут необхідно встановити, які об'єкти будуть існувати постійно, а які тимчасово - тільки на період виконання ними необхідних дій.

Коли об'єкти візуалізовані, можна приступати до специфікації повідомлень. При цьому слід враховувати ті ролі, які грають повідомлення в системі. При необхідності уточнення цих ролей треба використовувати їх різновиди та стереотипи. Для знищення об'єктів, які створюються на час виконання своїх дій, потрібно передбачити явне повідомлення.

Найбільш прості випадки розгалуження процесу взаємодії можна зобразити на одній діаграмі з використанням відповідних графічних примітивів. Однак слід пам'ятати, що кожен альтернативний потік управління може істотно ускладнити розуміння побудованої моделі. Тому загальним правилом є візуалізація кожного потоку управління на окремій діаграмі послідовності. У цій ситуації такі окремі діаграми повинні розглядатися спільно як одна модель взаємодії.

Подальша деталізація діаграми послідовності пов'язана з введенням тимчасових обмежень на виконання окремих дій в системі. Для простих асинхронних повідомлень тимчасові обмеження можуть бути відсутніми. Однак необхідність синхронізувати складні потоки управління, як правило, вимагають введення в модель таких обмежень. Загальна їх запис повинна слідувати семантиці мови об'єктних обмежень, який розглянуто в додатку.



*Рис.3. Діаграма послідовностей*

## Діаграма кооперацій

**Діаграма кооперації** призначена для специфікації структурних аспектів взаємодії. Головна особливість діаграми кооперації полягає в можливості графічно представити не тільки послідовність взаємодії, але й всі структурні відносини між об'єктами, що беруть участь у цій взаємодії.

Насамперед, на діаграмі кооперації у вигляді прямокутників зображуються об'єкти, що беруть участь у взаємодії, що містять ім'я об'єкта, його клас і, можливо, значення атрибутів. Далі, як і на діаграмі класів, вказуються асоціації між об'єктами у вигляді різних сполучних ліній. При цьому можна явно вказати імена асоціації й ролей, які грають об'єкти в даній асоціації. Додатково можуть бути зображені динамічні зв'язки - потоки повідомлень. Вони представляються також у вигляді сполучних ліній між об'єктами, над якими розташовується стрілка із вказівкою напрямку, імені повідомлення й порядкового номера в загальній послідовності ініціалізації повідомлень.

На відміну від діаграми послідовності, на діаграмі кооперації зображуються тільки відносини між об'єктами, що грають певні ролі у взаємодії. З іншого боку, на цій діаграмі не вказується час у вигляді окремого виміру. Тому послідовність взаємодій і паралельних потоків може бути визначена за допомогою порядкових номерів. Отже, якщо необхідно явно специфікувати взаємозв'язок між об'єктами в реальному часі, краще це робити на діаграмі послідовності.

Поводження системи може описуватися на рівні окремих об'єктів, які обмінюються між собою повідомленнями, щоб досягти потрібної мети або реалізувати деякий сервіс. З погляду аналітика або конструктора важливо представити в проекті системи структурні зв'язки окремих об'єктів між собою. Таке статичне подання структури системи як сукупності взаємодіючих об'єктів і забезпечує діаграма кооперації.

Таким чином, за допомогою діаграми кооперації можна описати повний контекст взаємодій як своєрідний тимчасовий "зрізу" сукупності об'єктів, взаємодіючих між собою для виконання певної задачі або бізнес-мети програмної системи. Кооперація може бути представлена на двох рівнях:

* на рівні специфікації - показує ролі класифікаторів і ролі асоціацій у розглянутій взаємодії.
* на рівні прикладів - указує екземпляри й зв'язки, що утворять окремі ролі в кооперації.

У кооперації рівня прикладів визначаються властивості, які повинні мати екземпляри для того, щоб брати участь у кооперації.

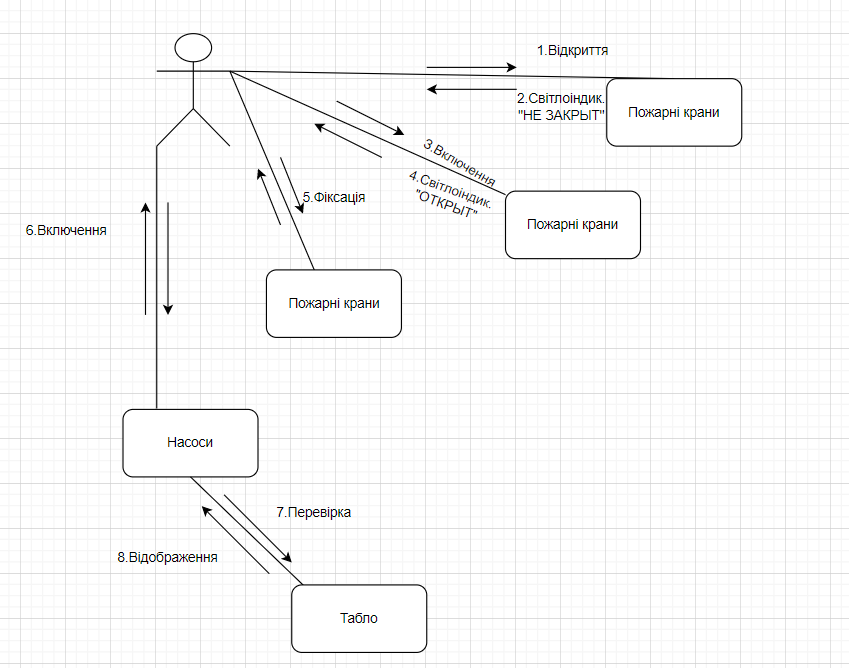
Об'єкт (object) є окремим екземпляром класу, який створюється на етапі виконання програми. Він може мати своє власне ім'я і конкретні значення атрибутів. Стосовно об'єктів формат рядка класифікатора доповнюється ім'ям об'єкта і набуває такого вигляду (при цьому вся запис підкреслюється): **<Ім'я об'єкта> '/' <Ім'я ролі класифікатора> ':' <Ім'я класифікатора>.**

У контексті мови UML всі об'єкти діляться на дві категорії: пасивні та активні. Пасивний об'єкт оперує тільки даними і не може ініціювати діяльність з управління іншими об'єктами, але можуть посилати сигнали в процесі виконання запитів, які вони отримують. Активний об'єкт (active object) має свою власну нитку (thread) управління і може ініціювати діяльність з управління іншими об'єктами.

Зв'язок (link) є екземпляром або прикладом довільної асоціації. Зв'язок як елемент мови UML може мати місце між двома і більше об'єктами. Поруч з лінією в її середній частині може записуватися ім'я відповідної асоціації. Зв'язки не мають власних імен, оскільки повністю ідентичні як екземпляри асоціації. Зв'язок може мати деякі стереотипи, які записуються поруч з одним з її кінців і вказують на особливість реалізації цих питань. У мові UML для цієї мети можуть використовуватися такі стереотипи:

* "Association" - асоціація (передбачається за замовчуванням, тому цей стереотип можна не вказувати).
* "Parameter" - параметр методу. Відповідний об'єкт може бути тільки параметром деякого методу.
* "Local" - локальна змінна методу. Її область видимості обмежена тільки сусіднім об'єктом.
* "Global" - глобальна змінна. Її область видимості поширюється на всю діаграму кооперації.
* "Self - рефлексивна зв'язок об'єкта з самим собою, яка допускає передачу об'єктом повідомлення самому собі. На діаграмі кооперації рефлексивна зв'язок зображується петлею у верхній частині прямокутника об'єкта.

Повідомлення на діаграмі кооперації специфікує комунікацію між двома об'єктами, один з яких передає іншому певну інформацію. При цьому перший об'єкт очікує, що після отримання повідомлення другим об'єктом піде виконання деякого дії. Таким чином, саме повідомлення є причиною або стимулом для початку виконання операцій, відправки сигналів, створення і знищення окремих об'єктів. Напрямок стрілки вказує на одержувача повідомлення.



*Рис.4. Діаграма кооперацій*

## Діаграми класів

**Діаграма класів** складається із множини елементів, які в сукупності відбивають декларативні знання про предметну область. Ці знання інтерпретуються в базових поняттях мови UML, таких як класи, інтерфейси й відносини між ними і їхніми складовими компонентами. При цьому окремі компоненти цієї діаграми можуть утворювати пакети для подання більше загальної моделі системи. Якщо діаграма класів є частиною деякого пакета, то її компоненти повинні відповідати елементам цього пакета, включаючи можливі посилання на елементи з інших пакетів.

Побудова діаграм класів (class diagrams) є центральною ланкою методології об’єктно-орієнтованого аналізу й проектування.

Діаграма класів відображає класи та їх взаємовідносини, тим самим представляючи логічний аспект проекту. Кожна діаграма класів представляє певний ракурс структури класів. На стадії аналізу діаграми класів використовуються, щоб виділити загальні ролі й обов'язки сутностей, які забезпечують необхідну поведінку системи. На стадії проектування діаграми класів застосовують, щоб передати структуру класів, які формують архітектуру системи.

Кожен клас повинен мати ім'я і, якщо ім'я занадто довге, його можна скоротити або збільшити сам значок на діаграмі. Ім'я кожного класу повинно бути унікальним в поточному проекті.

Діаграма класів визначає типи об'єктів системи й різного роду статичні зв'язки, які існують між ними. Є два основних види статичних зв'язків:

* асоціації (наприклад, менеджер може вести кілька проектів);
* підтипи (працівник є різновидом особистості).

На діаграмах класів зображуються також атрибути класів, операції та обмеження, які накладаються на зв'язки між об'єктами.

Фрагменти діаграми:

### Асоціації

Асоціації являють собою зв'язки між екземплярами класів (особистість працює в компанії, компанія має ряд офісів). У загальному випадку множинність вказує нижню й верхню границі кількості об'єктів, які можуть брати участь у зв'язку. Для цього можуть використовуватися однина, діапазон або дискретна комбінація із чисел і діапазонів.

Для асоціації може бути зазначений напрямок навігації. Якщо навігація зазначена тільки в одному напрямку, то така асоціація називається односпрямованою. У двонапрямленій асоціації навігація зазначена в обох напрямках. У мові UML відсутність стрілок в асоціації трактується наступним чином: напрямок навігації невідомий або асоціація є двонапрямленою.

### Атрибути

Атрибути багато в чому подібні до асоціацій. Різниця між ними полягає в тому, що атрибути припускають єдиний напрямок навігації – від типу до атрибута.

Залежно від ступеня деталізації діаграми, позначення атрибута може включати ім'я атрибута, тип і значення за замовчуванням. У синтаксисі UML це виглядає так: <ознака видимості> <ім'я> : <тип> = <значення за замовчуванням>, де ознака видимості може набувати одне з чотирьох значень:

* загальний (public) - атрибут доступний для всіх клієнтів класу;
* захищений (protected) - атрибут доступний тільки для підкласів і друзів класу;
* секретний (private) - доступний тільки для друзів класу;
* реалізація (implementation) - атрибут доступний тільки усередині пакета, що обрамляє.

### Операції

Операції являють собою процеси, реалізовані класом. Найбільш очевидна відповідність існує між операціями й методами класу. Повний синтаксис UML для операцій виглядає так: <ознака видимості> <ім'я> (<список-параметрів>) : <тип виразу, що повертає значення> = <рядок-властивостей>, де:

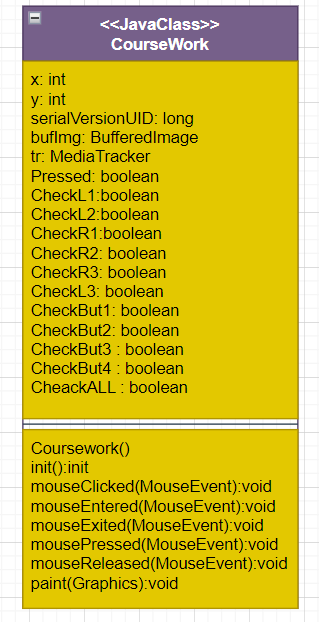
* ознака видимості може набувати ті ж значення, що й для атрибутів;
* ім'я являє собою символьний рядок;
* список параметрів містить необов'язкові аргументи, синтаксис яких збігається із синтаксисом атрибутів;
* тип виразу, що повертає значення, є необов'язковою специфікацією й залежить від конкретної мови програмування;
* рядок властивостей показує значення властивостей, які застосовуються до даної операції.

### Узагальнення

Узагальнення з погляду реалізації пов'язане з поняттям успадкування в мовах програмування. Підклас успадковує всі методи й поля суперкласу і може перевизначити наслідувані методи. Підтип можна також реалізувати, використовуючи механізм делегування.

### Обмеження

При побудові діаграм класів основним завданням є відображення різних обмежень. За допомогою конструкцій асоціації, атрибута й узагальнення можна специфікувати найбільш важливі обмеження, але неможливо виразити їх усі. В UML відсутній чіткий синтаксис опису обмежень, за винятком поміщення їх у фігурні дужки {}.



*Рис.5. Діаграма класів*

## Діаграми станів

Діаграми станів (state diagrams) є добре відомим засобом описання поведінки систем. Вони визначають усі можливі стани, в яких може перебувати конкретний об'єкт, а також процес зміни станів об'єкта в результаті впливу деяких подій.

**Діаграма станів** по суті є графом спеціального виду, що представляє деякий автомат. Поняття автомата в контексті UML має досить специфічну семантику, заснованої на теорії автоматів. Вершинами цього графа є стани й деякі інші типи елементів автомата, які зображуються відповідними графічними символами. Дуги графа служать для позначення переходів зі стану в стан. Діаграми станів можуть бути вкладені друг у друга, створюючи вкладені діаграми більше детального подання окремих елементів моделі. Для розуміння семантики конкретної діаграми станів необхідно представляти не тільки особливості поводження модельованої сутності, але й знати загальні відомості по теорії автоматів. У мові UML під станом розуміється абстрактний метаклас, використовуваний для моделювання окремої ситуації, протягом якої має місце виконання деякої умови. Стан може бути задане у вигляді набору конкретних значень атрибутів класу або об'єкта, при цьому зміна їхніх окремих значень буде відбивати зміна стану модельованого класу або об'єкта.

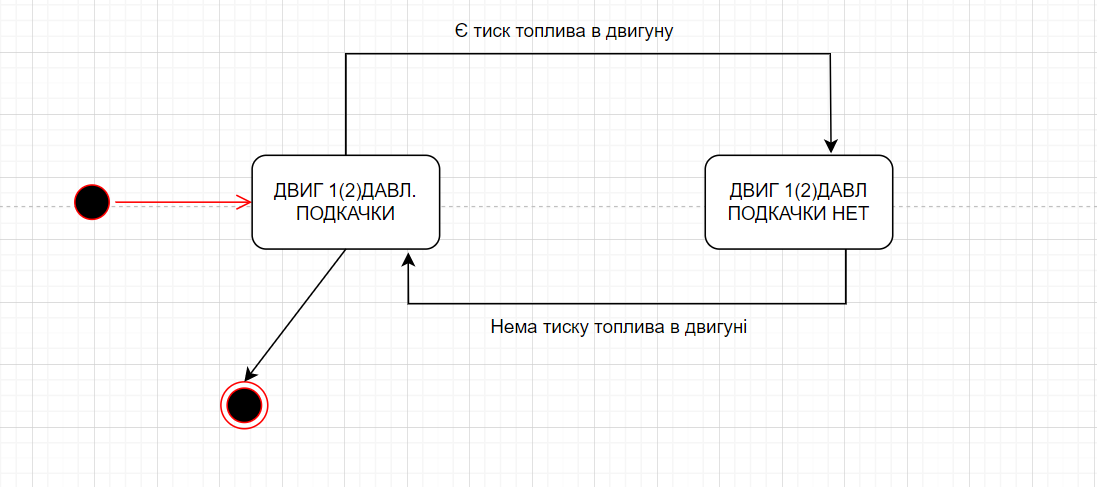
*Ім'я стану* являє собою рядок тексту, що розкриває змістовний зміст даного стану. Ім'я завжди записується із заголовної букви.

*Початковий стан* являє собою окремий випадок стану, що не містить ніяких внутрішніх дій. У цьому стані перебуває об'єкт за замовчуванням у початковий момент часу. Воно служить для вказівки на діаграмі станів графічної області, від якої починається процес зміни станів. Кінцевий (фінальний) стан являє собою окремий випадок стану, що також не містить ніяких внутрішніх дій. У цьому стані буде перебувати об'єкт за замовчуванням після завершення роботи автомата в кінцевий момент часу. Воно служить для вказівки на діаграмі станів графічної області, у якій завершується процес зміни станів або життєвий цикл даного об'єкта.

*Простий перехід (simple transition)* являє собою відношення між двома послідовними станами, що вказує на факт зміни одного стану іншим.

Перехід здійснюється при настанні деякої події: закінчення виконання діяльності (do activity), одержанні об'єктом повідомлення або прийомом сигналу. На переході вказується ім'я події . Об'єкт перейде з одного стану в інше в тому випадку, якщо відбулася зазначена подія й сторожова умова прийняла значення "істина".

*Сторожова умова (guard condition),* якщо воно є, завжди записується в прямих дужках після події-тригера і являє собою деяке булевські вираження. Нагадаємо, що булевські вираження повинне приймати одне їх двох взаємно, що виключають значень: "істина" або "неправда". Якщо сторожова умова приймає значення "істина", то відповідний перехід може спрацювати, у результаті чого об'єкт перейде в цільовий стан. Якщо ж сторожова умова приймає значення "неправда", то перехід не може спрацювати, і при відсутності інших переходів об'єкт не може перейти в цільовий стан по цьому переході.



*Рис.6. Діаграма станів*

# Програмні засоби

Програмна ралізація даного курсового проекту була виконана за допомогою такої мови програмування як Java, а саме Java-аплета. **Java** широко відома як новітня об'єктно-орієнтована мова, легка у вивченні і це дозволяє створювати програми, які можуть виконуватися на будь-якій платформі без якихнебудь доопрацювань (кросплатформеність). Також Java орієнтована на Internet, і найпоширеніше її застосування - невеликі програми, аплети, які запускаються в браузері і є частиною HTML-сторінок.

**Java** — об'єктно-орієнтована мова програмування, випущена компанією Sun Microsystems у 1995 році як основний компонент платформи Java. Синтаксис мови багато в чому походить від C та C++. Java програми виконуються у середовищі віртуальної машини Java. Java програми компілюються у байткод, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. В 2009 році Sun Microsystems придбала компанія Oracle, яка продовжує розвивати Джаву.

Закінчена, надзвичайно надійна і універсальна технологія Java неоціненна для розробників програмного забезпечення тим, що вона дозволяє їм:

* Писати програмне забезпечення на одній платформі і запускати його практично на будь-який інший.
* Створювати програми для роботи у веб-браузерах і веб-службах
* Розробляти серверні додатки для інтернет-форумів, магазинів, опитувань, обробки HTML-форм і ін.
* Створювати максимально пристосовані для вимог користувача програми та служби, комбінуючи різні Java-модулі.
* Писати потужні та ефективні програми для мобільних телефонів, віддалених процесорів, недорогих споживчих товарів і практично для будь-яких цифрових пристроїв.

Java є повністю об’єктно-орієнтованою мовою, тому потрібно засвоїти принципи ООП якомога краще.

Насправді доволі часто різні автори по різному виділяють головні концепції ООП. Причиною є те, що в ООП справді доволі багато понять. І з плином часу та розвитком програмування їх лише більшає. Різноманітні поняття доволі тісно взаємопов'язані між собою. В літературі часто виділяють наступні три концепції і навіть вказують, що вони основні для ООП ("три кити"):

* **інкапсуляція** (incapsulation) - концепція побудови класів через закриття(капсулювання) їхньої реалізації.
* **успадкування** (inheritance) - створення одних класів на основі інших
* **поліморфізм** (polymorphism) - можливість використання батьківських класів замість класів нащадків. По суті є частиною реалізованої в мові концепції успадкування.

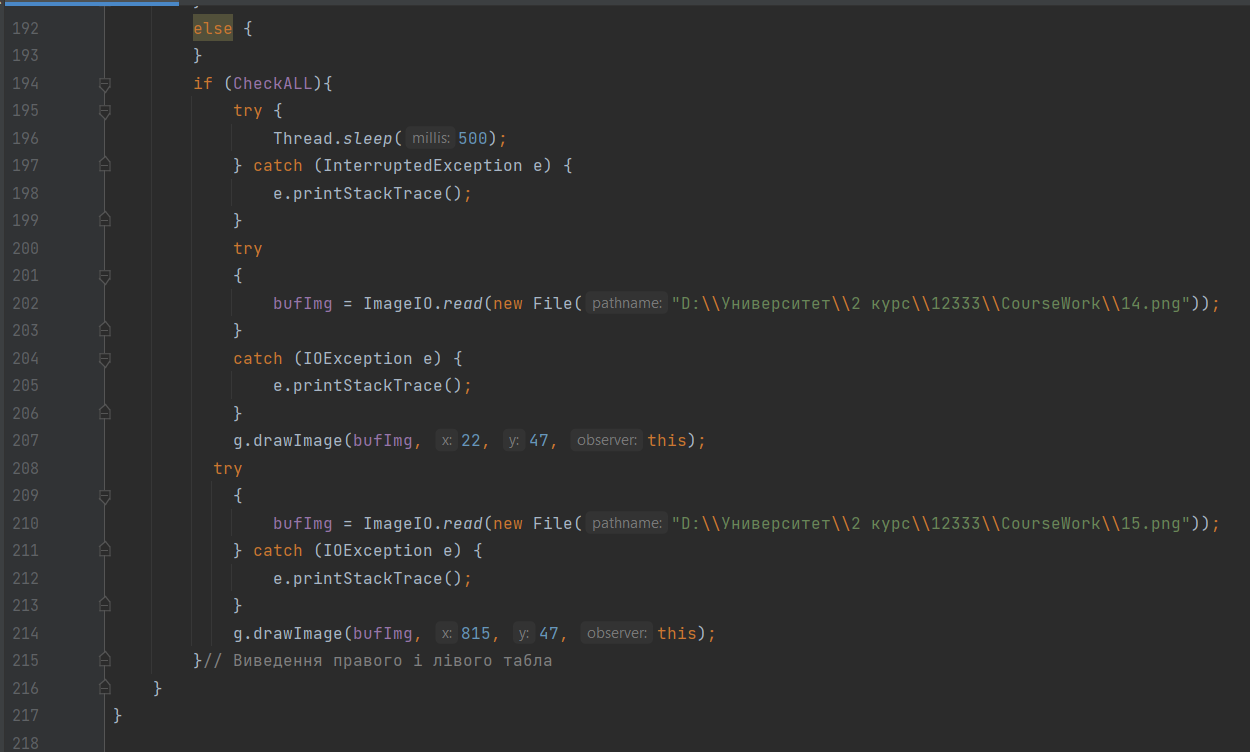
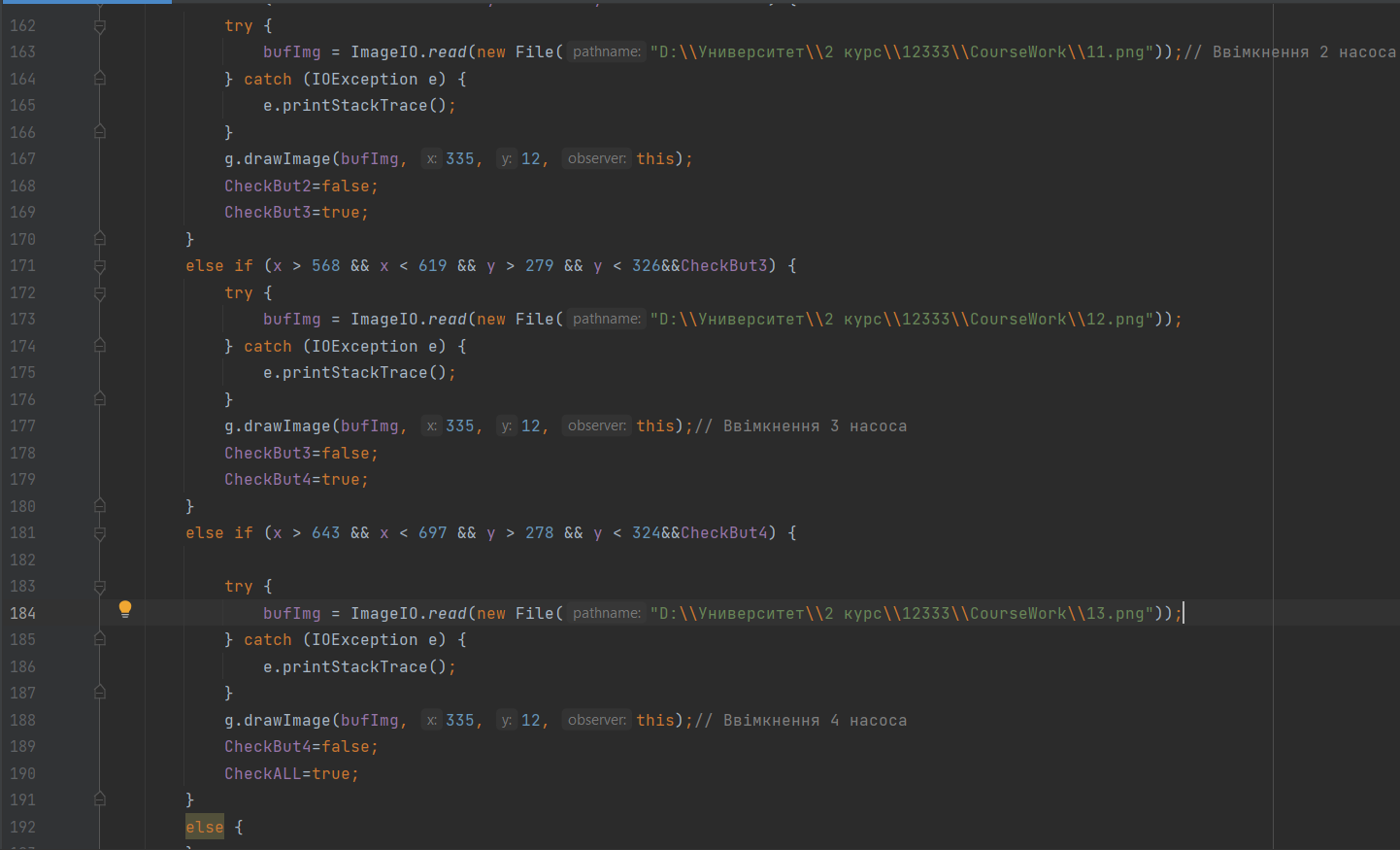
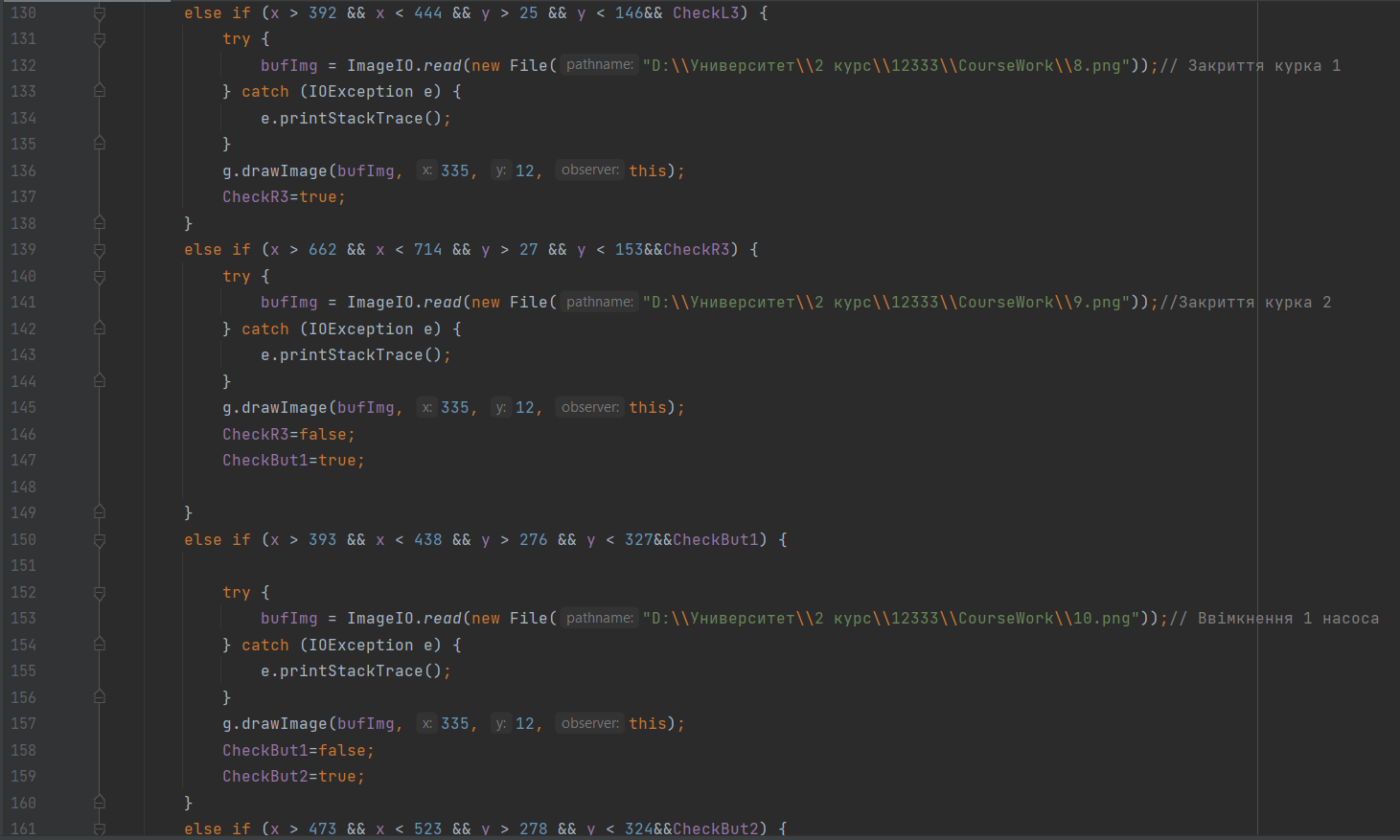
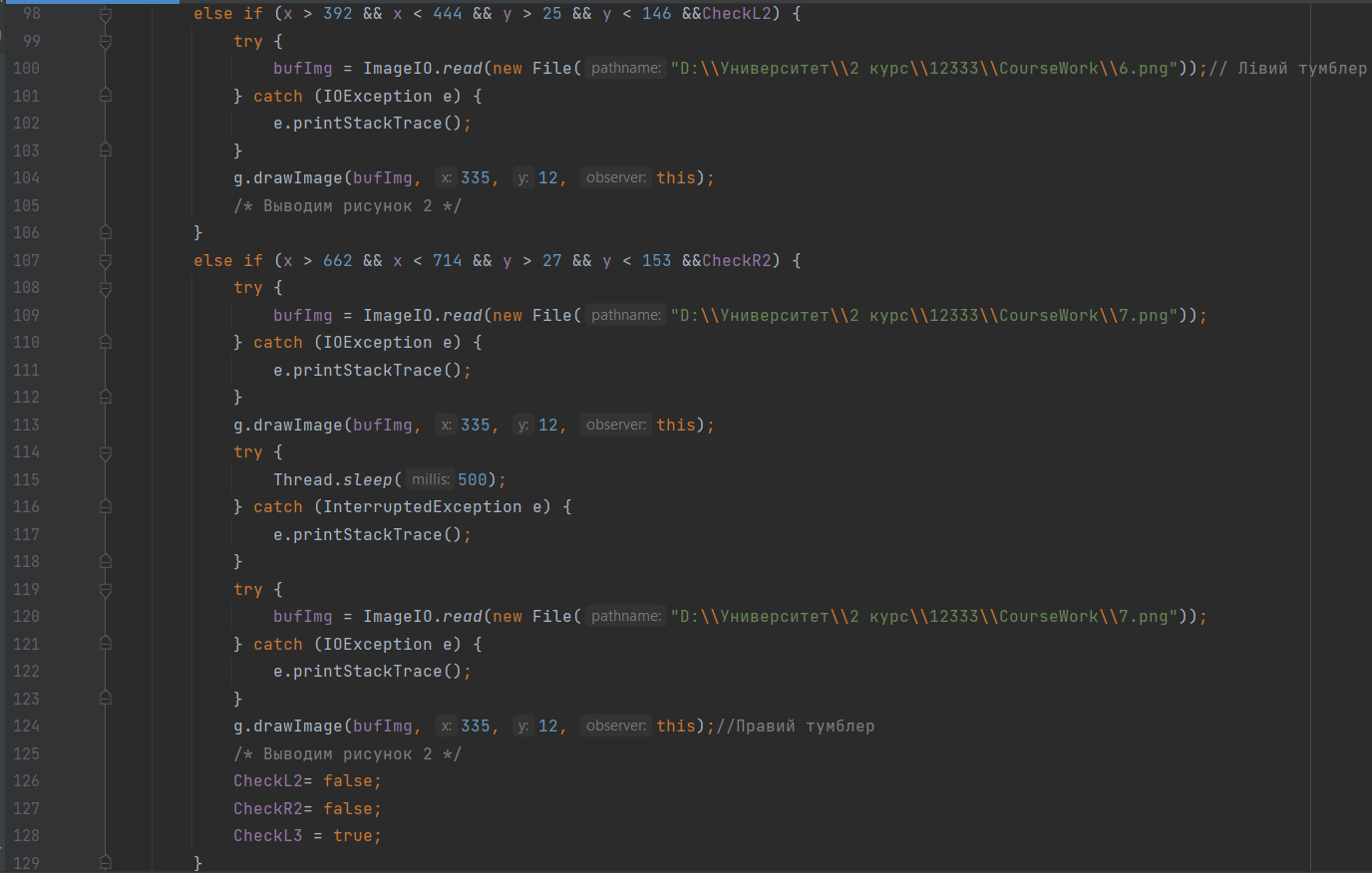
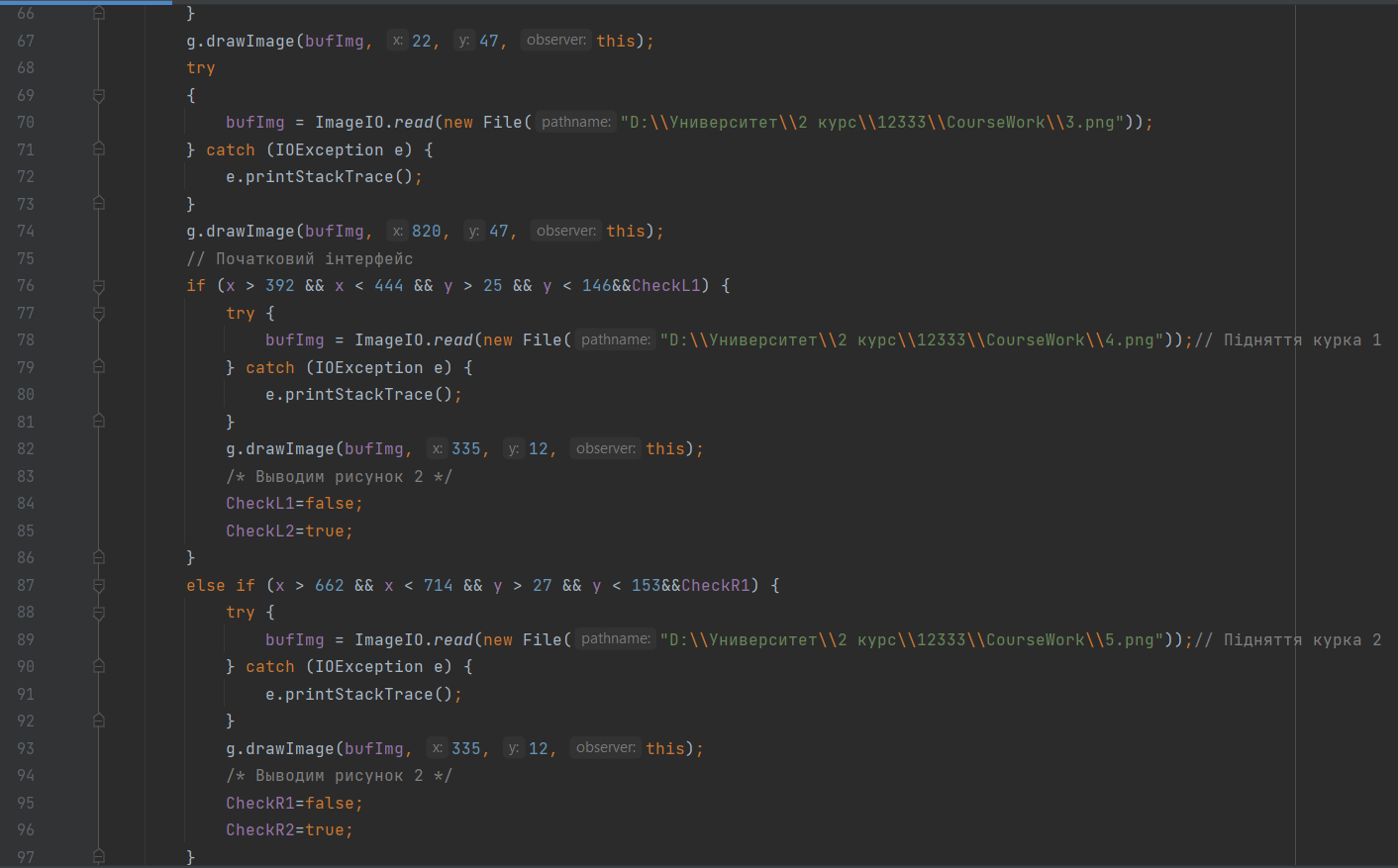
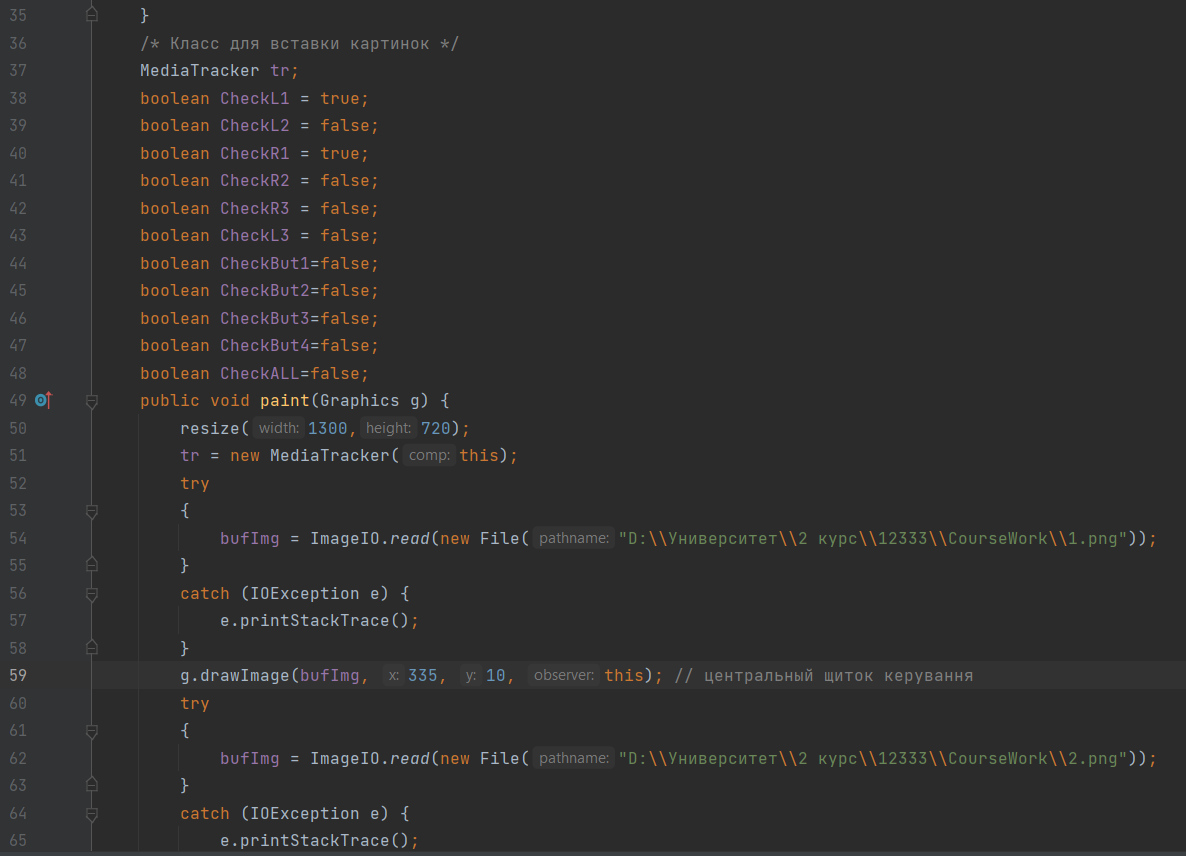
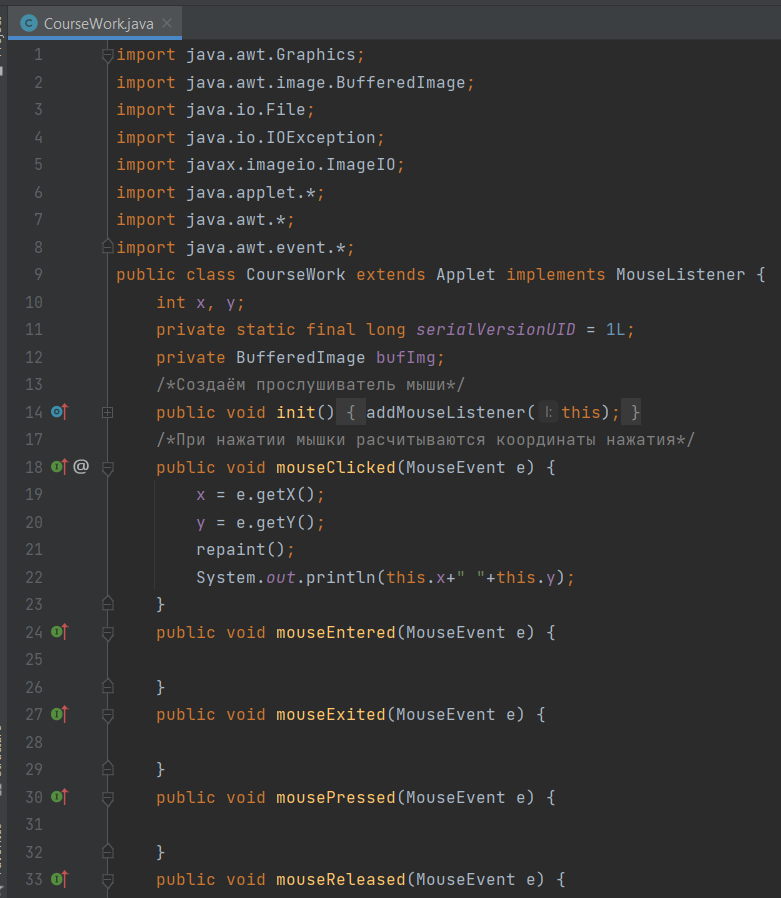
**Java-аплет** — прикладна програма, яка запускається при кожному звертанні до Web-сторінці, у вихідний текст якої вона вбудована або переноситься програма на Java в формі байт-коду, поширювана через Webсторінки.

Аплет — це невелика програма на мові Java, яка може виконуватися у вікні Інтернет-браузера. Однак, аплет може бути і велика, складна програма, взаємодіюча, наприклад, з програмою на сайті, звідки цей аплет був завантажений. Одне з типових використань аплетів — це ігри на сайтах.

Java[-](https://uk.wikipedia.org/wiki/Java)аплет забезпечує можливість використовувати в World Wide Web

«виконуваний вміст» шляхом виконання в гіпертекстовому документі мініпрограм, відомих як аплети. Незвичайним є формат готового до виконання коду, у який транслюються аплети Java, — це апаратно-незалежний побайтовий формат. Завдяки використанню такого нейтрального формату відкомпільовані Java-програми можна переміщати по Internet[,](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) оскільки вони не залежать від операційноїсистеми, у якій виконуються.

# 6 Програмна реалізація



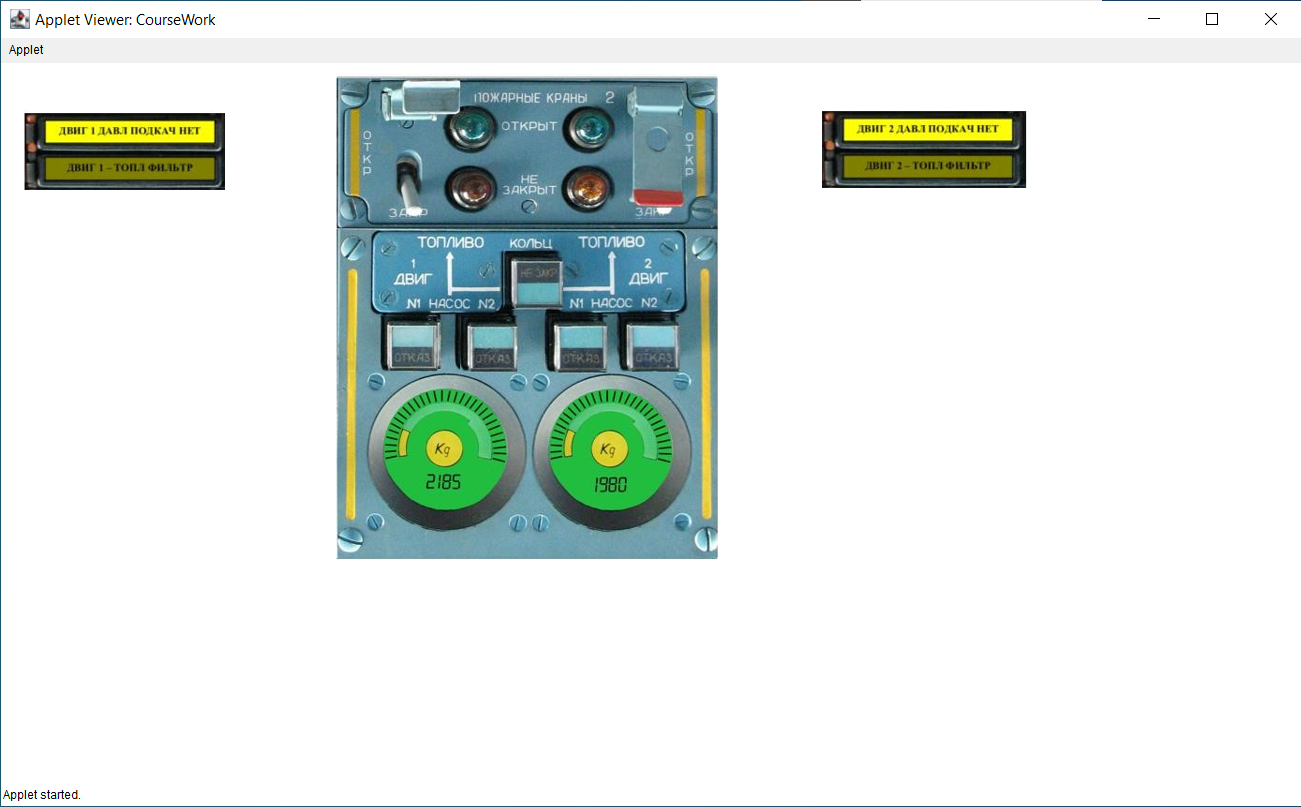
**7. Результат виконання програми**

У результаті виконання програмного коду ми отримуємо зображення бортової системи (рис.7).

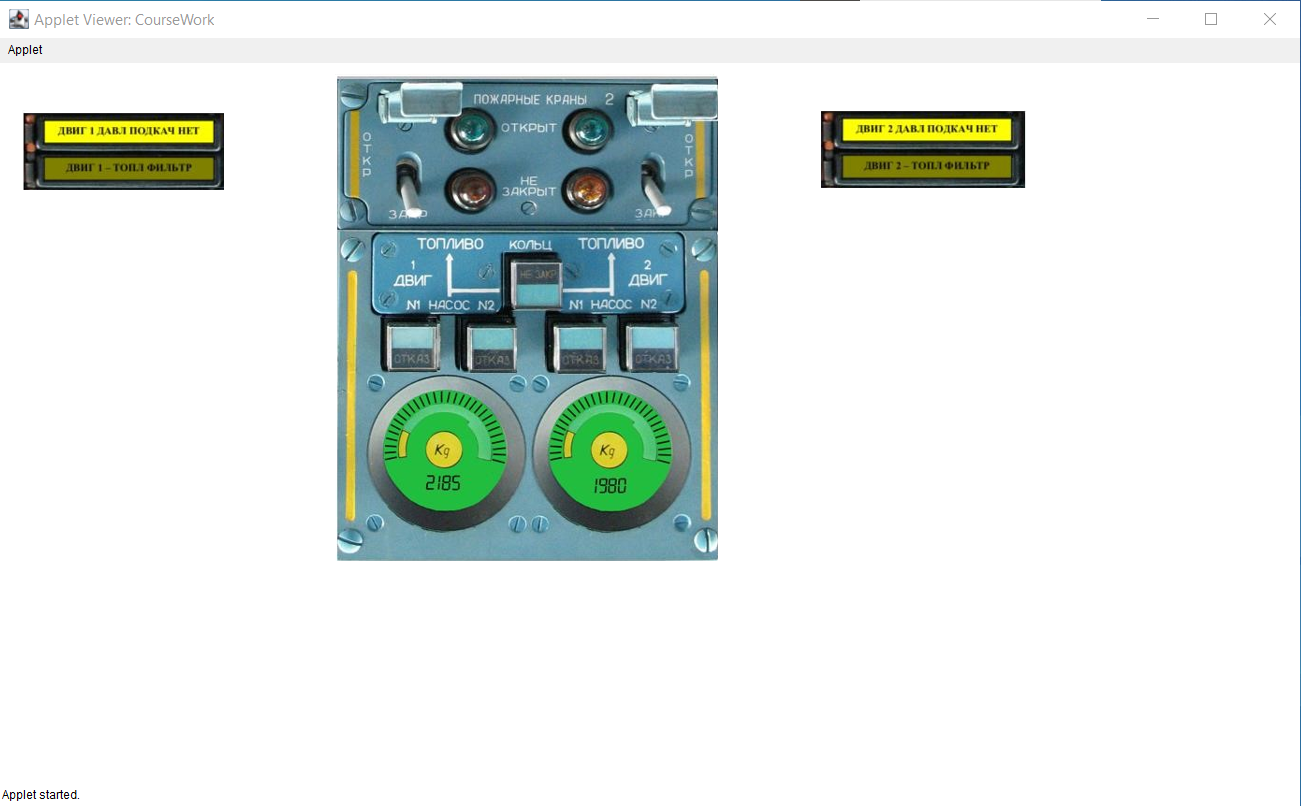


*Рис. 7 Старт запуску програми*

Відкриваємо пожежні крани двигунів, встановлюючи на щитку «ПОЖАРНЫЕ КРАНЫ» два перемикачі в верхнє положення. (рис.8, 9).



*Рис. 8. Програмний інтерфейс підняття курків*



*Рис. 9. Програмний інтерфейс підняття курків*

Вмикаємо пожежні крани, після цього короткочасно загораються світлоіндикатори «НЕ ЗАКРЫТ» та засвітяться світлоіндикатори «ОТКРЫТ»

(рис.10,11).

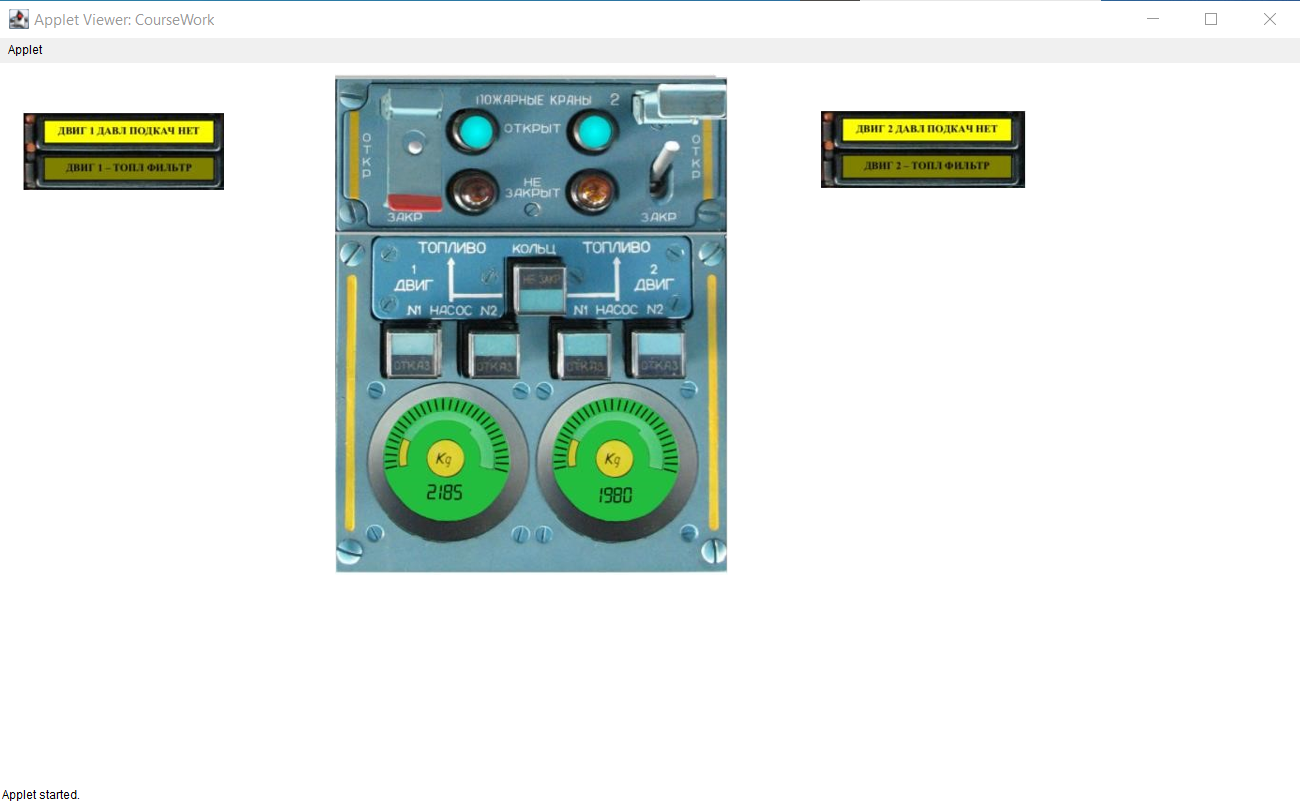


*Рис. 10. Програмний інтерфейс вмикання тумблерів*



*Рис. 10.б Програмний інтерфейс вмикання тумблерів*

Після того як відкрили крани, зафіксовуємо тумблери курками (рис.11,12).



*Рис. 11. Програмний інтерфейс зафіксованого тумблера (лівий)*



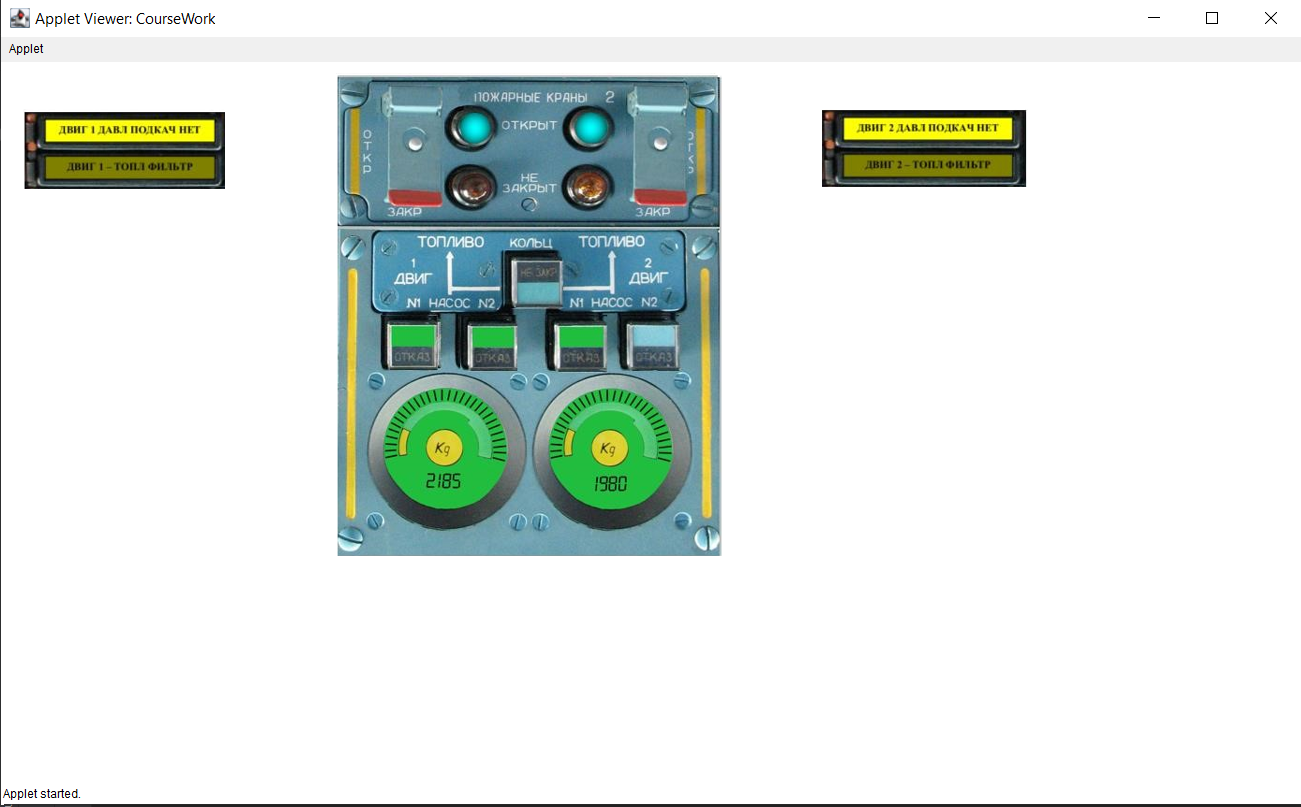
*Рис. 12. Програмний інтерфейс зафіксованого тумблера (правий)*



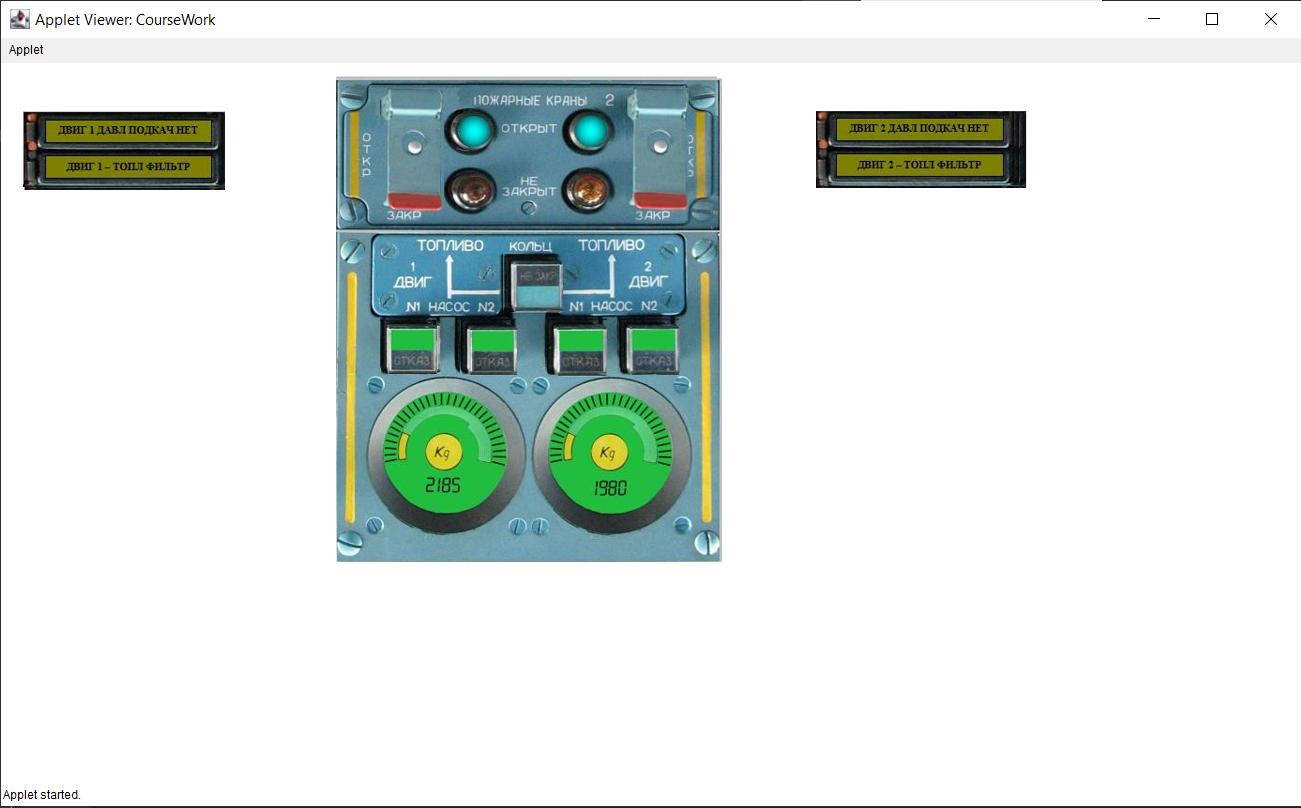
*Рис. 13. Програмний інтерфейс ввімкнутого першого насоса*



*Рис. 14. Програмний інтерфейс ввімкнутого другого насоса*

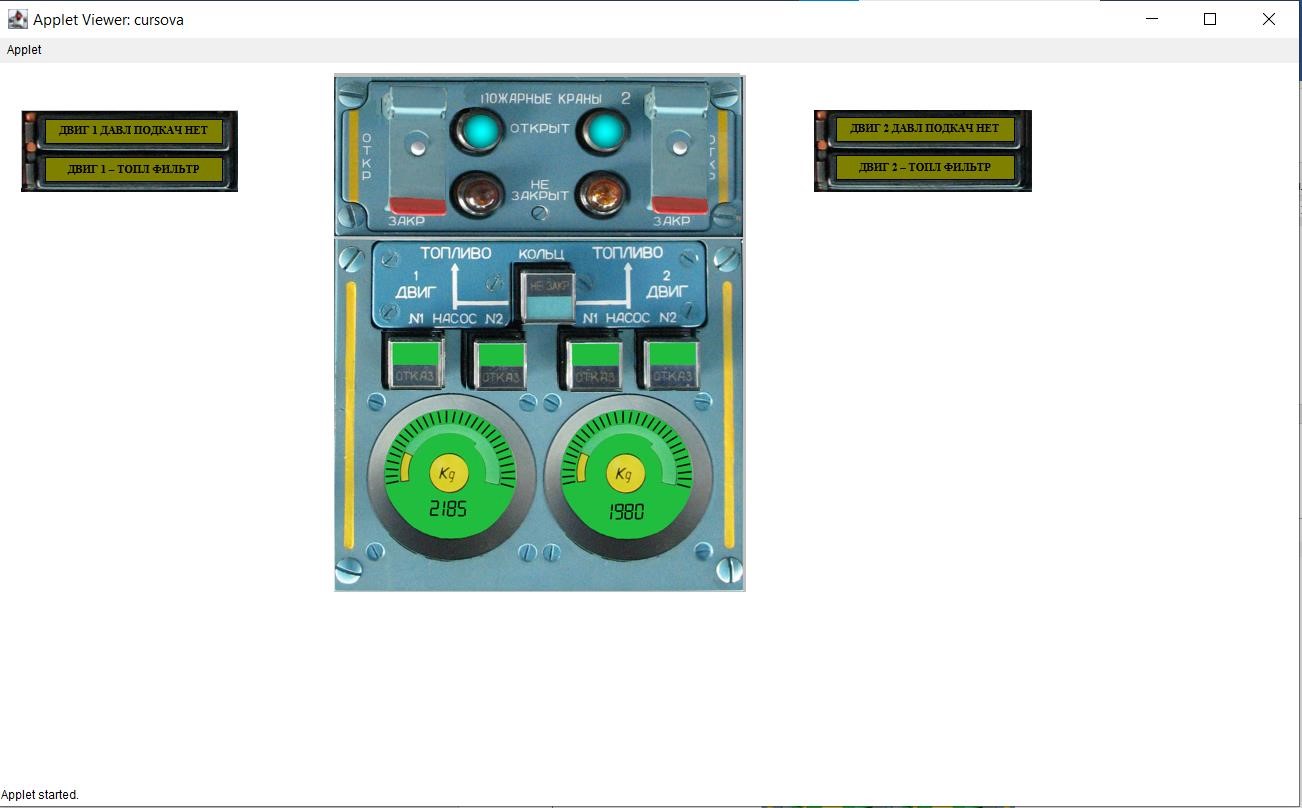


*Рис. 15. Програмний інтерфейс ввімкнутого третього насоса*



*Рис. 16. Програмний інтерфейс ввімкнутого четвертого насоса*

Після ввімкнення четвертого насоса, погаснуть табло «ДВИГ1 ДАВЛ ПОДКАЧ НЕТ» і «ДВИГ2 ДАВЛ ПОДКАЧ НЕТ» (рис.17)



*Рис. 17. Програмний інтерфейс погаслого табло «ДВИГ1 ДАВЛ ПОДКАЧ НЕТ» і «ДВИГ2 ДАВЛ ПОДКАЧ НЕТ»*

**8. Висновок**

В результаті виконання даної курсової роботи нами була розроблена об’єктна модель системи «Засоби контролю та управління паливною системою літака АН-140» та змодельовано процес її функціонування, а саме нами була реалізована взаємодія оператора з паливною системою літака АН-140 на етапі перевірки кількості палива у баках.

А також звикористанням мови візуального моделювання UML були побудовані діаграми:

* варіантів використання (Use Case Diagram);
* класів (Class Diagram);
* станів (Statechart Diagram);
* послідовності (Sequence Diagram);
* кооперацій (Collaboration Diagram);

Програмну реалізацію було виконано із застосуванням мови програмування високого рівня Java.

**9. Список літератури**

1. Райчев І.Е., Харченко О. Г., Замковий В.В. Принципи проектування відкритих розподілених систем: // Навчальний посібник . - К.: НАУ, 2010. - 240 с.
2. Леоненков А. Самоучитель UML. Серия "Самоучитель". - СПб.: BHVСанкт-Петербург, 2002 - 304 с.
3. Буч, Гради. Объектно-ориентированный анализ и проектирование. Пер. с англ. 2-е изд. М.: «Бином». 2000.-560с.
4. Кватрани Е. Rational Rose 2000 и UML. Визуальное моделирование. 2001.

176с.

5. [Електронный ресурс] Вікіпедія. Діаграма прецедентів. Режим доступу: [https://uk.wikipedia.org/wiki/Діаграма\_п](https://uk.wikipedia.org/wiki/Діаграма)рецедентів