ВИМОГИ ДО КУРСОВОГО ПРОЄКТУ

1. Загальні умови:

- мова програмування (C#, Python) – на вибір студента.
- програмний продукт – віконний (graphical user interface) додаток.
- вхідні дані подаються користувачем на GUI.
- результати відображаються користувачу у віконному режимі.
- сховище даних – це текстовий файл, тому передбачити збереження даних у файл та зчитування даних з файлу.
- передбачити навігаційну панель (меню), яка повинна забезпечувати перегляд, редагування, додавання, видалення даних.
- передбачити сортування, фільтрацію та пошук даних по самостійно вибраному ключу.
- обов'язкова перевірка коректності вводу даних та обробка виключень.
- наявність «посібника користувача» – інструкції з використання програми – це окрема вкладка в пункті меню (до якого підключено окремий UI екран, простими словами форму), яку розгорнувши можна прочитати як користуватись вашою програмою: що натиснути і т.д.

2. Умови програмної реалізації

Програмна реалізація має складатися не менше ніж з 4-5 класів, серед яких передбачити класи-нащадки, абстрактні класи та класи-інтерфейси.

Крім завдань, що вказані в індивідуальній умові, передбачити реалізацію не менше 5-ти «власно-придуманих» методів згідно предметної області.

Всі класи повинні містити:

- конструктор для ініціалізації об'єкта;
- конструктор перезавантаження із параметрами – значення створюваних об'єктів вводяться з клавіатури і передаються в конструктори об'єктів у вигляді параметрів;
- деструктор для звільнення пам'яті (з повідомленням про знищення об'єкта);
- методи обробки даних, зазначені в індивідуальному завданні.

Всі поля класу зробити закритими (модифікатор доступу *private*), а доступ, ініціалізацію та зміну кожного поля виконати через властивості (C#: *get/set,* Python: функція *property*).

В класах-нащадках передбачити поліморфізм методів класу.

РОЗРОБКА ПРОГРАМИ

1. Загальні правила

При розробці індивідуального завдання курсового проєкту використати об'єктно-орієнтований підхід, який завдяки принципам інкапсуляції, успадкування та поліморфізму дає можливість виділити всі спільні та відмінні риси окремих елементів поставленого завдання і записати їх у програмі в найлаконічнішій формі.

Розробка програми при об'єктно-орієнтованому підході розбивається на розробку класів та глобальних типів даних, реалізацію методів, глобальних функцій, створення об'єктів та організацію взаємодії між ними в головній програмі для досягнення поставленої мети.

2. Розробка системи класів

Після того, як виконана декомпозиція програми (розроблена її структура) і відомі алгоритми розв'язку задач, можна приступати до реалізації наміченого плану:

- Потрібно чітко уявити, з якими даними буде працювати програма (чи окремий модуль) та які дії над ними доведеться виконувати. Дані та дії над ними утворюють об'єкт.

Також можна уявити програму як набір об’єктів (реально існуючих фізичних об’єктів або абстрактних понять, що існують лише в нашій уяві), які взаємодіють між собою.

– Наступним етапом буде поділ об’єктів на групи, що мають хоча б щось спільне (дані, властивості, функціональність). У більшості випадків об’єкти будуть, більш чи менш, подібні між собою. Деколи в групах можна виділити підгрупи і т.д., аж до складного ієрархічного дерева. У кожній групі виділяється найзагальніші властивості, притаманні всім без винятку об’єктам у групі та дії над ними. Отже, отримаємо базовий клас для цієї групи об’єктів, даними якого будуть загальні параметри об’єктів групи, а методами – дії, які можна проводити над даними будь-якого об’єкта групи.

– Далі для кожної з підгруп (у межах виділеного класу об’єктів) додаємо притаманні лише їй властивості, як поля нового, породженого від базового класу. Додаткові дії, які властиві кожній підгрупі, стають методами породжених класів, а параметри, яких не мали об’єкти базового класу, – даними породжених (похідних) класів. Породження класів продовжується до тих пір, поки не будуть описані всі параметри об’єктів та дії, які необхідно над ними здійснювати для розв’язку поставлених задач.

– Описана послідовність повторюється для кожної з Групи об’єктів у програмі.

Для прикладу розглянемо програму, яка повинна продемонструвати гру в шахи. У ній можна виділити такі дві Групи об’єктів: перша – реалізує графічне відображення на екрані шахової дошки та фігур на ній, друга – реалізує логіку роботи програми – генерує ходи, перевіряє їх правильність, слідкує за часом і т.д.

Розглянемо першу групу:

1) З чого вона складається?

Сюди входять різноманітні видимі елементи: дошка, клітинки на ній, різні фігури, позначення рядів на дошці.

2) Які в них параметри?

Це – координати, розміри, колір для фігур, для клітинки – ще тип фігури, яка на ній знаходиться, для дошки – ще масив клітинок на ній і т.д.

3) Які дії повинні виконувати об’єкти?

Вони повинні реагувати або не реагувати на натискання мишкою чи при настиканні відповідної клавіші клавіатури, відображатися, зникати, деякі ще повинні переміщуватись, причому по-різному.

4) Що ж є спільним для них усіх?

Серед параметрів – напевне, координати центра, ще колір і розмір по горизонталі та вертикалі, з дій – кожен із об’єктів повинен уміти створюватися, знищуватися та відображати себе на екрані. Таким чином, базовий клас буде містити такі поля даних: координати «х», «у», «ширина», «висота», «колір»; методи – конструктор, що створює об’єкт, деструктор, що його знищує, та метод «відобрази себе».

Оце й усе! Але описане лише один раз (!) для всіх видимих об’єктів.

5) Далі можна розділити видимі об’єкти на пасивні та активні.

Пасивними будуть написи на дошці, що позначають назви рядів, активними – всі решта, вони повинні мати метод, який змушує їх реагувати на натискання мишкою. Тепер виділимо ще підклас фігур, що породжується від класу «активні видимі елементи», який має додатковий метод «перемісти мене» та поле даних або метод, що визначає допустимі ходи для даної фігури. Так, дошка містить 64 клітинки, а кожна клітинка – може містити якусь фігуру. Тому до даних цього класу, також породженого від «активних видимих елементів», слід додати ще поле даних (напевне, вказівник), який містить інший об’єкт(-и) класу – «видимі елементи». Також необхідні методи, які ці об’єкти «вставляють» у клас «Група» або видаляють з класу «Група». Метод «відобрази себе» для «Група» завдяки поліморфізму можна доповнити відображенням об’єктів «Група» після того, як відобразилася сама «Група». Далі від «Група» можна успадкувати нові класи: «Шахова дошка», «Шахове поле», а від «Шахового поля» – «Поле останнього рядка», яке крім методів свого попередника може ще й перетворювати «пішака» у «королеву».

Цей приклад, далеко не повний, він дає можливість зрозуміти лише основний підхід до проєктування класів. Пофантазувавши трохи, можна отримати завершену ієрархію об’єктів, які будуть уміти робити все, що від них може вимагати головна програма для розв’язування поставлених задач. Основне – кожна спільна для кількох об’єктів дія (і спільний елемент даних) описана лише один раз у базовому класі і більше ніде її не потрібно описувати повторно.

Етап «Розробка системи класів» відображає основну роботу, яка виконується при об’єктно-орієнтованому програмуванні, в результаті чого, отримуються описи всіх класів з даними та методами, причому для методів уже задані їх параметри та тип результату, який вони повертають. Тобто, з точки зору інтерфейсу, робота над класами на цьому етапі вже завершена. Залишилося тільки «навчити» методи виконувати свою роботу. Глобальні функції, які не є методами ні одного з класів, також поміщаються в один із модулів, а їх інтерфейсна частина описується в цьому підрозділі.

3. Розробка методів

На даному етапі слід описати принцип дії основних методів та відмінності поліморфних методів від методів базових класів. Це завдання є досить простим, якщо на попередньому етапі було правильно спроєктовано систему класів та детально пророблено інтерфейсні частини.

Реалізація будь-якої функції (чи методу) – запис тих дій, які має виконувати метод чи функція, не забуваючи користуватися вже готовими функціями та методами замість того, щоб увесь час змінювати дані за допомогою одних і тих же послідовностей операцій.

Так, метод «перемісти мене» (див. приклад у попередньому пункті) повинен лише змінити координати активного видимого елемента, а не намагатися перемалювати його самостійно – метод «відобрази себе» зробить це набагато краще. Аналогічно метод «відобрази себе» для «групи» (шахової дошки чи клітинки), який повинен відображати об’єкти «групи» після того, як відобразилася сама група, реалізується з використанням поліморфізму та успадкування. При цьому додається лише один-два оператори, оскільки і група, і видимі елементи вже самі вміють відображатися, необхідно лише «сказати» їм, щоб вони це зробили один за одним. У методі «відобрази себе» групи спочатку викликається метод «відобрази себе» базового класу («видимий елемент»), який відображає саму дошку або клітинку, а далі – методи «відобрази себе» для всіх видимих елементів, що входять до групи. Для клітинки – це буде виклик «відобрази себе» для фігури, що стоїть на клітинці, а для дошки – виклик «відобрази себе» для всіх позначок та клітинок, які, у свою чергу, відобразять фігури, що на них знаходяться.

4. Створення об’єктів і розробка головної програми

На даному етапі слід описати процес створення самих об’єктів у програмі, функціональність та дані яких розроблені на попередніх етапах. Якщо кількість об’єктів, що будуть створені наперед невідома, або обсяг пам’яті, яку вони займають, є досить значним, тоді пам’ять для об’єктів слід виділяти «динамічно».

Стосовно розглянутого прикладу із шахами, даний етап буде описувати – як створюється шахова дошка з 32 фігурами на ній та як перетворюються вибрані партнерами ходи в повідомлення шаховим фігурам. Створивши об’єкти, можемо реалізувати певну послідовність дій або схему взаємодії між об’єктами, яка призводить до вирішення поставленого завдання. Опис цієї послідовності дій чи схеми взаємодії, можливо, з прикладами для реальних вхідних даних чи повідомлень і є основою даного етапу. Такий опис повинен бути достатнім для розуміння всіх деталей реалізації алгоритму та можливих випадків, що зустрічаються при роботі програми.

5. Опис файлів даних та інтерфейсу програми

Якщо програма використовує файли як джерело вхідних даних або для зберігання проміжних чи кінцевих результатів роботи, то на даному етапі слід навести опис формату цих файлів.

Особливо важливий даний етап для програм, що працюють із базами даних, адже структура таблиць бази даних, перелік, типи полів, засоби взаємодії з базою даних, перетворення даних є основою таких програм. Для інтерактивної програми з розвинутою системою меню та діалогових вікон у цьому етапі слід описати призначення елементів меню, роботу з ними, параметри, що вибираються в діалогових вікнах тощо. Тут же слід описати інтерфейс програм, які працюють з параметрами командного рядка.

6. Вимоги до програми

Незалежно від теми програма, що розробляється, повинна задовольняти наступним загальним вимогам.

– ***Стійкість програми***. Програма не повинна втрачати працездатності ні при яких, навіть некоректних, діях користувача. Всякі дії, що загрожують втратою інформації, мають бути підтверджені користувачем. Інформація, що вводиться, скрізь, де це можливо, піддається логічному забезпеченню цілісності даних. При будь-яких діях користувача не повинні втрачатися дані або їх цілісність (некоректність індексів, втрата посилань в зв'язках після видалення-додавання записів і т. д.).

– ***Функціональна повнота***. Мають бути реалізовані усі функції (методи), вказані в специфікації програми.

– ***Інтерфейс користувача***. Використовуються тільки терміни, зрозумілі користувачеві, і не використовуються терміни розробника («запис», «індексація» і т.д.). Поява службових англомовних повідомлень неприпустима. У повідомленнях користувача слід дотримуватися норм ввічливості, колірна гамма повинна наслідувати загальноприйняті рекомендації.

– ***Використання клавіатури.*** На будь-якому етапі натиснення будь-якої клавіші повинне ігноруватися або викликати передбачені дії, описані в засобах допомоги. Прив'язка дій до клавіш має бути загальноприйнятою: F1 – допомога; Enter – згода, завершення введення; Esc – відмова, повернення до попереднього вузла гілки алгоритму (з відновленням екранної форми); Tab – перехід до наступного поля, вікна і т.д.; Shift & Tab – повернення до попереднього поля, вікна і т.д.