Лекція 1

**Занальні положення дисципліни**

План.

1. Загальна концепція людинно-машиної взаємодії. Поняття інтерфейсу користувача.
2. Види користувацьких інтерфейсів.
3. Оцінка ефективності інтерфейсу за швидкістю його роботи
4. Відділення інтерфейсу від моделі.
5. MVC - шаблон.

**Загальна концепція людинно-машиної взаємодії**

На сьогоднішній день розвиток інформаційних технологій та створення штучного інтелекту займає вагоме місце у наукових дослідженнях та роботах багатьох вітчизняних та іноземних дослідників і науковців. Взаємодія людини і комп’ютера є однією з визначальних особливостей існування в сучасному світі. В наш час постала необхідність не просто оволодіння і накопичення інформації, а й навчання такій технології роботи з інформацією, яка дозволить вилучати інформацію з різних джерел, представляючи її в нормальному вигляді, формі та форматі і яка в подальшому буде ефективно використовуватися для прийняття рішень в тій чи іншій сфері проектування та розробки людино-машинної взаємодії.

Сучасні системи штучного інтелекту і нейрокомп’ютерні системи неможливо порівняти зі здібностями людини в плані інтелекту і творчих здібностей. Водночас безсумнівні заслуги технічних засобів у звільненні людини від рутинних операцій, в розширенні її часових ресурсів для вирішення творчих завдань. Ось чому в даний час важливо не ставити під сумнів можливості технічних засобів і не покладати необґрунтовані надії на безмежне зростання цих можливостей, а найкращим чином використовувати наявні резерви штучних інформаційних систем і людини для вирішення практичних завдань.

В даний час стає все більш очевидно, що найбільш перспективний шлях підвищення якості інформаційних процесів в людській діяльності, соціумі, ноосфері – спільне використання можливостей людини і програмно-технічних засобів, зокрема, інформаційних систем.

Людино-машинна взаємодія - дисципліна, що має справу з розробкою, розвитком і застосуванням комп'ютерних систем з акцентом на користувацький інтерфейс. Є два аспекта, в яких може бути розглянуто користувацький інтерфейс – взаємодія із користувачем та взаємодія із функціональною частиною програмного забезпечення (моделлю). Почнемо із загального розгляду першого із них.

**Поняття інтерфейсу користувача.**

Інтерфе́йс користувача́ — засіб зручної взаємодії користувача з інформаційною системою. Сукупність засобів для обробки та відбиття інформації, якнайбільше пристосованих для зручності користувача; у графічних системах інтерфейс користувача, втілюється багатовіконним режимом, змінами кольору, розміру, видимості (прозорість, напівпрозорість, невидимість) вікон, їхнім розташуванням, сортуванням елементів вікон, гнучкими налаштуваннями як самих вікон, так і окремих їх елементів (файли, теки, ярлики, шрифти тощо), доступністю багатокористувацьких налаштувань.

Під інтерфейсом користувача (КІ) програми розуміємо сукупність елементів, що дозволяють користувачу програми управляти її роботою і отримувати необхідні результати.

Інтерфейс користувача — це простір, де відбувається співпраця між людьми та машинами. Мета цієї співдії, полягає у тому, щоби забезпечити досконалу роботу та керування машиною з боку людини, а машина одночасно, надає інформацію, яка допомагає прийняттю рішень операторами. Конструктивні міркування, що застосовуються під час створення інтерфейсів користувача, пов'язано з такими галузями, як ергономіка та психологія.

Як правило, мета створення інтерфейсу користувача, полягає у тому, щоби зробити користувацький інтерфейс, який спрощує (самозрозумілий), ефективний і приємний (зручний для користувача) для керування машиною таким чином, щоби забезпечити бажаний результат. Це, зазвичай, означає, що оператор повинен застосовувати щонайменші зусилля для досягнення очікуваного підсумку, а також, щоби машина зменшувала небажані результати для людини. З посиленням використання персональних комп'ютерів та відносним зниженням обізнаності (інтересу) суспільства про важкі машини, термін «інтерфейс користувача», як правило, передбачає графічний інтерфейс користувача, тоді як промислові панелі керування та механізми контролю за обладнанням, частіше стосуються (HMI) людино-машинної взаємодії.

**Види користувацьких інтерфейсів**

Апаратні інтерфейси - це фізичні, просторові інтерфейси, присутні на виробах у повсякденному житті від тостерів, до приладових панелей автомобілів чи кабін літаків. Вони, як правило, є поєднанням ручок, кнопок, слайдерів, перемикачів і сенсорних екранів.

Інтерфейси командного рядка, де користувач вводить вхідний сигнал, введенням командного рядка за допомогою комп'ютерної клавіатури, а система забезпечує виведення шляхом друку тексту на моніторі комп'ютера. Використовується програмістами та системними адміністраторами в інженерних і наукових середовищах, а також технічно просунутими користувачами персональних комп'ютерів.

Пакетні інтерфейси є неінтерактивними користувацькими інтерфейсами, де користувач заздалегідь задає всі деталі пакетного завдання для пакетної обробки і отримує висновок, коли всю обробку виконано. Комп'ютер не запитує подальше введення, після початку обробки.

Графічні користувацькі інтерфейси (GUI) приймають вхідні дані за допомогою таких пристроїв, як комп'ютерна клавіатура та миша, й забезпечують графічний висновок на моніторі комп'ютера. У GUI-дизайні, широко використовуються як мінімум, два різні принципи: об'єктно-орієнтовані користувацькі інтерфейси (OOUI) й інтерфейси, орієнтовані на додатки.

Веб-інтерфейси користувача або веб-користувацькі інтерфейси (WUI), які приймають вхідні дані та забезпечують виведення, створенням веб-сторінок, які передаються Інтернетом і проглядаються користувачем за допомогою програми веб-браузера. Адміністративні веб-інтерфейси для веб-серверів, серверів і мережевих комп'ютерів, часто називаються панелями керування.

Сенсорні екрани - це дисплеї, які приймають введення дотиком пальців або стилусом. Використовується у все більшій кількості мобільних пристроїв і багатьох видах точок продажу, промислових процесах і машинах, машинах самообслуговування тощо.

Розмовні інтерфейси дозволяють користувачам керувати комп'ютером за допомогою: звичайного тексту (наприклад, за допомогою текстових повідомлень або чатів), чи голосових команд замість графічних елементів. Ці інтерфейси часто наслідують людські розмови.

Інтерфейси жестів є графічні користувацькі інтерфейси, які приймають вхідні дані у формі жестів руки або миші, накреслених за допомогою комп'ютерної миші або стилуса.

Голографічні користувацькі інтерфейси забезпечують вхід для електронних або електромеханічних пристроїв, передаванням пальця через відтворені голографічні зображення того, що в іншому випадку було б тактильним контролем цих пристроїв, що вільно плавають у повітрі, виявленим джерелом хвилі та без тактильної взаємодії.

Провівши деяке узагальнення, можна виділити наступні 3 напрямки:

1 ) командний інтерфейс - користувач дає команди комп'ютеру , який їх виконує і видає результат користувачеві. Командний інтерфейс реалізований у вигляді пакетної технології та технології командного рядка;

2 ) WIMP -інтерфейс ( WIMP від : Window - вікно ; Image - образ ; Menu - меню; Pointer - покажчик ) - діалог користувача з комп'ютером ведеться за допомогою графічних образів: меню , вікон та інших елементів . Інтерфейс реалізований на двох рівнях технологій : простий графічний інтерфейс і WIMP - інтерфейс;

3 ) SILK -інтерфейс ( SILK від : Speak - розмова ; Image - образ ; Lenguage - мова ; Knowlege - знання) - розмова користувача з комп'ютером . Інтерфейс найбільш наближений до звичайної , людської формі спілкування . При цьому комп'ютер визначає команди, аналізуючи людську мову і знаходячи в ній ключові фрази. Результат виконання команд комп'ютер перетворює в зрозумілу людині форму. Цей вид інтерфейсу найбільш вимогливий до апаратних ресурсів комп'ютера.

**Оцінка ефективності інтерфейсу за швидкістю його роботи**

Існують 4 основні критерії якості будь-якого інтерфейсу:

* швидкість роботи користувачів;
* кількість людських помилок;
* швидкість навчання;
* суб'єктивне задоволення;

Всі інші критерії - похідні, наприклад, такі як зрозумілість, лаконічність, ефективність і т.д. Всі подібні критерії є лише складовими цих чотирьох критеріїв.

Далі розглянемо тільки перший критерій. Багато користувачів оцінюють інтерфейс (особливо використовуваний дуже часто) тільки з позиції швидкості роботи, тобто чим швидше він працює, тим краще. Тривалість виконання роботи користувачем складається з тривалості інтелектуальної роботи, тривалості фізичних дій користувача і тривалості реакції системи. Як правило, тривалість реакції системи є найменш значущим фактором, який практично ніяк не залежить від користувача.

Час, що витрачається на виконання інтелектуальної роботи, ми також обчислити не можемо ні за яких умов, так як кожен користувач сприймає різну інформацію зі своєю швидкістю. Тому розглянемо тільки час, що витрачається на фізичні дії користувача.

Для порівняльної оцінки користувальницьких інтерфейсів використовується модель GOMS.

**Спрощена модель GOMS для проведення порівняльної оцінки користувальницьких інтерфейсів**

GOMS - модель, заснована на оцінці швидкості друку. Розробники моделі GOMS під час її створення помітили, що час, потрібний для виконання якогось завдання системою «користувач-комп'ютер», є сумою всіх тимчасових інтервалів, які потрібні були системі на виконання послідовності елементарних жестів, що складають дану задачу.

Операції, що входять в модель GOMS

K = 0.2 с. Натискання клавіші. Час, необхідний для того, щоб натиснути клавішу.

P = 1.1 c. Вказівка. Час, необхідний для того, щоб вказати на якусь позицію на екрані монітора.

H = 0.4 с. Переміщення. Час, необхідний користувачу для того, щоб перемістити руку з клавіатури на ГУВ (графічний пристрій введення) або з ГУВ на клавіатуру.

М = 1.35 с. Ментальна підготовка. Час, необхідний користувачу для того, щоб розумово підготуватися до наступного кроку.

R. Відповідь. Час, протягом якого користувач повинен очікувати відповідь комп'ютера.

**Правила розстановки ментальних операцій**

Правило 0. Початкова розстановка оператора М: Оператори М слід встановлювати перед усіма операторами К (натискання клавіші), а також перед усіма операторами P (вказівка ​​за допомогою ГУВ), призначеними для вибору команд; але перед операторами P, призначеними для вказівки на аргументи цих команд, ставити оператор М годі було.

Правило 1. Видалення очікуваних операторів М: Якщо оператор, наступний за оператором М, є повністю очікуваним з точки зору оператора, що передує М, то цей оператор М може бути видалений.

Наприклад, якщо ви переміщаєте ГУВ з наміром натиснути його кнопку після досягнення мети руху, то відповідно до цього правила слід видалити оператор М, який встановлюється за правилом 0. У цьому випадку послідовність P M K перетворюється в P K.

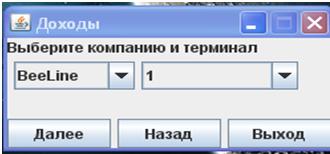
Правило 2. Видалення операторів М всередині когнітивних одиниць: Якщо рядок виду М К М К М К ... Належить когнітивної одиниці, то слід видалити всі оператори М, крім першого. Когнітивної одиницею є безперервна послідовність символів, що вводять, які можуть утворювати назву команди або аргумент. Наприклад «Y», «переміщати», «4,556» є прикладами когнітивних одиниць.

Правило 3. Видалення операторів М перед послідовними роздільниками: Якщо оператор До означає зайвий розділовий знак, який стоїть в кінці когнітивної одиниці (наприклад, роздільник команди, наступної відразу за роздільником аргументу цієї команди), то слід видалити оператор М, що стоїть перед ним.

Правило 4. Видалення операторів М, які є переривниками команд: Якщо оператор До є роздільником, що стоять після постійної рядка (наприклад, назва команди або будь-яка послідовність символів, яка кожного разу вводиться в незмінному вигляді), то слід видалити оператор М, що стоїть перед ним . (Додавання роздільника стане звичним дією, і тому роздільник стане частиною рядка і не буде вимагати спеціального оператора М.) Але якщо оператор До є роздільником для рядка аргументів або будь-який інший змінною рядки, то оператор М слід зберегти перед ним.

Правило 5. Видалення перекривають операторів М: Будь-яку частину оператора М, яка перекриває оператор R, що означає затримку, пов'язану з очікуванням відповіді комп'ютера, враховувати не слід.

Розглянемо конкретний приклад розрахунку часу за допомогою моделі GOMS:



Для початку складемо послідовність дій, які виконуються в даному вікні інтерфейсу. Вона буде включати в себе вибір компанії і терміналу. Потім натискання кнопки «Далі».

Отримаємо послідовність: PKPKPKPKPK.

Далі за правилом 0 ставимо перед кожними P і K, операцію М: MPMKMPMKMPMKMPMKMPMK.

За правилом 1, PMK переходить в PK: MPKMPKMPKMPKMPK.

Останні дві операції P і K, припускають вибір і натискання кнопки «Далі». Так як це цілком очікуване дію, то останню М ми також прибираємо. Після чого в підсумку отримаємо: MPKMPKMPKMPKPK.

Так як інші правила не застосовуються, то взявши дані нам значення часу виконання операцій, отримаємо Т = 11.8 сек.

Насправді модель GOMS дозволяє обчислити час лише приблизно. Так як для кожного користувача час виконання будь-яких операцій своє. Тобто досвідчений користувач може друкувати швидше, ніж новачок. Крім цього варто враховувати, що будь-яка фізична дія, що здійснюється користувачем, може бути або точним або швидким. Пояснюється це суто фізіологічними факторами: при різкому русі неможливо швидко зупинитися, відповідно, чим точніше має бути рух, тим більше плавним і уповільненим воно повинно бути. Таким чином, щоб фізична дія користувача було швидким, воно не повинно бути точним. Користувач, як правило, керує комп'ютером двома способами, а саме мишею і клавіатурою. Клавіатура не вимагає особливої ​​точності рухів - неважливо, швидко натиснули кнопку або повільно, так само як сильно або слабо. Миша, навпаки, інерційна - є різниця між повільним її переміщенням і швидким, сильним прикладеним зусиллям і слабким. Саме тому оптимізація використання миші в системі може істотно підвищити загальну швидкість роботи.

**Закон Фітса**

У 1954 році Пол Фітс сформулював правило:

«Час досягнення мети обернено пропорційно розміру мети і дистанції до неї.»

Тобто використовуючи це правило, стає зрозуміло, що вказівка ​​мишею буде залежати від розміру кнопки і відстані до неї. Тому час операції P можна обчислити за формулою:, де a - середнє час запуску / зупинки руху, b - величина, що залежить від типової швидкості руху, D - відстань від точки старту до центру цілі, W - ширина цілі, виміряна вздовж осі руху.

Виходячи із закону Фітс, можна «нескінченно» прискорити натискання кнопки. Це можна зробити двома способами, зробити її нескінченно великою, або зробити нульовий дистанцію до неї.

Кнопка нескінченного розміру. При підведенні курсору до краю екрану він зупиняється, навіть якщо рух миші триває. Це означає, що кнопка, розташована впритул до верхнього або нижнього краю екрану, має нескінченну висоту (так само як кнопка біля лівого або правого краю має нескінченну ширину). Таким чином, швидкість досягнення такої кнопки залежить тільки від відстані до неї (ну і точності вибору початкового напрямку руху). Зрозуміло, що кнопка, розташована в кутку екрану, має «ще більш нескінченні» розміри, якщо так взагалі можна сказати (тобто не важливо навіть, з якою точністю переміщали миша). Для досягнення такої кнопки від користувача потрібно всього лише смикнути мишу в потрібному напрямку, не піклуючись про її швидкості і не роблячи спроб зупинити її в потрібному місці.

Нульова дистанція до кнопки забезпечується за допомогою контекстного меню. Наприклад при натисканні правої кнопки миші «випливе» діалогове вікно, яка розташовується впритул до курсора. Тим самим дистанція до кнопок цього вікна відразу стає мінімальною.

Тим самим щоб підвищити швидкість роботи інтерфейсу, варто виконувати деякі умови:

* Часто використовувані кнопки зробити найбільш помітними і досить великого розміру.
* Також можна часто використовувані кнопки розташувати в діалоговому вікні, що викликається правою кнопкою миші.
* У полях введення повинні стояти найбільш імовірні значення. (В цьому випадку користувачеві, якщо дане значення його задовольнить, залишається лише підтвердити введення).

Звичайно насправді подібних умов набагато більше. Але саме ці дозволять скоротити час роботи, за рахунок операції вказівки курсором на кнопку.

**Відділення інтерфейсу від моделі.**

Система може мати кілька користувацьких інтерфейсів для обслуговування різних користувачів. Наприклад, комп'ютеризована бібліотечна база даних, може містити два користувацьких інтерфейси, один для керівників бібліотек (обмежений набір функцій, прилаштований для простоти використання), а інший для працівників бібліотеки (широкий набір функцій, пристосованих для продуктивності).

Також може бути реалізован декілька користувацький інтерфейсів за типом взаємодії (графічний, командний, голосовий) та за типом користувацького пристрою (desktop, web, мобільний).

Дана особливість передбачає виконання одних і тих же програмних функцій у різний спосіб і в різних місцях програмного коду, одже можу привести до дублювання програмного коду. Крім того, це може внести додаткому плутанину в рограмних код, оскільки деякі елементи різних інтерфейсів можуть дублюватися не в повному обсязі або об’єднувати в собі декілька команд (копіювання декількох файлів через графічний інтерфейс, голосові команди, що пеедбачають відкриття додатку та виконання певної дії, та ін).

Іншим аспектом незручності розробки та супроводження ПЗ може бути наявність в одному місці програмного коду функціональних та інтерфейсних блоків, що призводить для важкості впровадження змін в коді. Додатково супроводження програмних систем ускладнюється наявністю групової розробки, при чому найчастіше моделлю та інтерфейсом займаються різні розробники. Навіть при розділенні моделі на інтерфейсу можуть виникнути проблеми із роботою одного із компонентів при зміні іншого, при умові їх прямого контактування одного із іншим.

Розглянемо перераховані незручності на прикладах:

Приклад 1. Задача збереження та редагування номера телефона. Інтерфейс командного рядка.

public class PhoneNumber {

private void getPhone() {

// 1. Read data from file

// 2. Show phone number

}

private void setPhone() {

// 1. Ask to enter phone number

// 2. Read phone number to the variable

// 3. Write phone number to the file

}

public void run() {

while (true) {

String command = "e";

// 1. Ask to enter command (read, write or exit)

if (command == "read") {

getPhone();

} else if (command == "write") {

setPhone();

} else {

break;

}

}

}

}

Приклад 2. Додавання графічного інтерфейсу.

public class PhoneNumber {

private void getPhone() {

// 1. Read data from file

// 2. Show phone number

}

private void setPhone() {

// 1. Ask to enter phone number

// 2. Read phone number to the variable

// 3. Write phone number to the file

}

private void runCommand() {

while (true) {

String command = "e";

// 1. Ask to enter command (read, write or exit)

if (command == "read") {

getPhone();

} else if (command == "write") {

setPhone();

} else {

break;

}

}

}

private void showForm() {

// 1. Drow dialog window with text box and button

// 2. Read data from file

// 3. Show phone number in text box

// 4. Bound event onBtnClick with method setPhone2()

}

private void writePhone() {

// 1. Get phone number from the text box

// 2. Write phone number to the file

}

public void run(String arg) {

if (arg == "commandUI") {

runCommand();

} else {

showForm();

}

}

}

Приклад 3. Додавання специфічної функції «зміна коду оператора».

private void setOperatorCode() {

// 1. Ask to enter code

// 2. Read code to the variable

// 3. Read phone number file

// 4. Change phone number

// 5. Write phone number to the file

}

private void runCommand() {

while (true) {

String command = "exit";

// 1. Ask to enter command (read, write or exit)

if (command == "read") {

getPhone();

} else if (command == "write") {

setPhone();

} else if (command == "change code") {

setOperatorCode();

} else {

break;

}

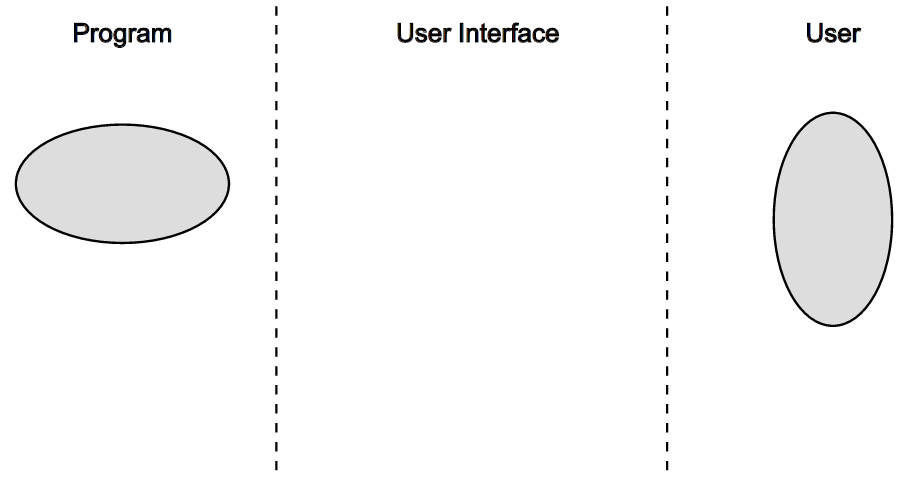
}

}

Як бачмо, програмний код для вирішення досить простої задачі стає досить складним (довгим, важко зрозумілим та складним у супроводженні). Можна уявити, що буде ыз кодом при додаванні нових інтерфейсів, функцій програми, чи наприклад можливсті збереження телефону в іншому форматі (наприклад у БД). Для вирішення вищенаведених проблем може бути застосовано підходи шаблонного проектування, а саме шаблон MVC.

**MVC – шаблон**

Якщо у нас є користувач, значить повинен бути призначений для користувача інтерфейс. Що ж таке інтерфейс? Це суміжна межа між двома системами. У нашому випадку: з одного боку - програма, з іншого - користувач. Ось вони.



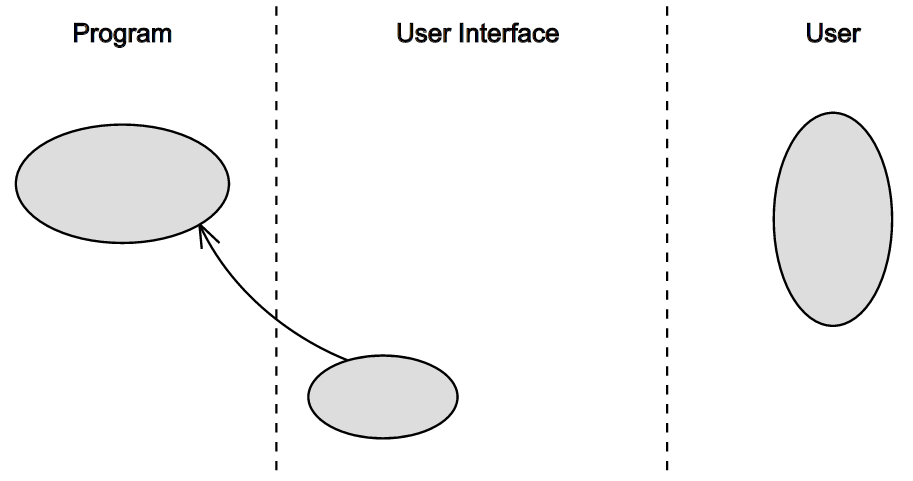
Програма абсолютно абстрактна, будь який предметний код. Вона вміє робити щось корисне, і у користувача є потреби, які можна задовольнити за допомогою цієї програми. Тоді з'являються шматочки логіки, які «знають», як, використовуючи цю програму, зробити безпосередньо те, що хоче користувач. Шматочки - не предметні, предметна логіка в програмі. Вони більше відносяться до користувача з його конкретними потребами, і являють собою комбінації викликів і звернень до програми.

Як приклад уявіть термінал для торгівлі на біржі. Користувач терміналу виставляє заявку, в якій вказує, що він хоче купити акції компанії «Світлий шлях» в кількості 20 штук за ціною 1500 за акцію. Також вказує, що заявка дійсна протягом чотирьох годин, і з якого з його рахунків списати гроші, в разі успішної угоди.

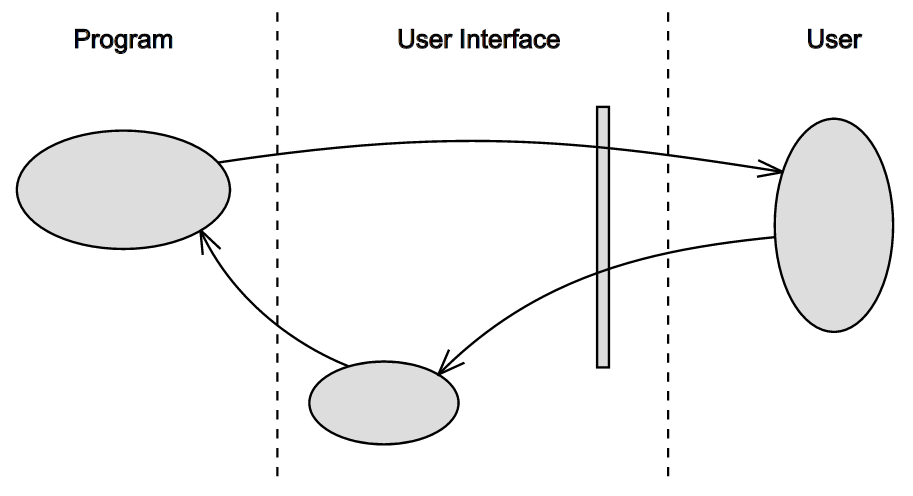
Відчутна кількість атрибутів. Проходить деякий час, і він розуміє, що за такою ціною купити не вдасться і готовий підняти ціну до 1550, залишивши всі інші значення. Тоді він вибирає цю заявку, натискає кнопку «змінити», вказує нову ціну, так. Це зручно.

Але на біржі не можна змінити заявку, в предметної області немає такого поняття. Заявку можна тільки виставити і скасувати. Щоб дати користувачеві можливість в один клік міняти заявку, треба запам'ятовувати старі значення, знімати заявку, давати редагувати те, що запам'ятали, і виставляти нову заявку. Така комбінація. Але для користувача вона виглядає як одну просту дію: зміна заявки. Це називається - use case.

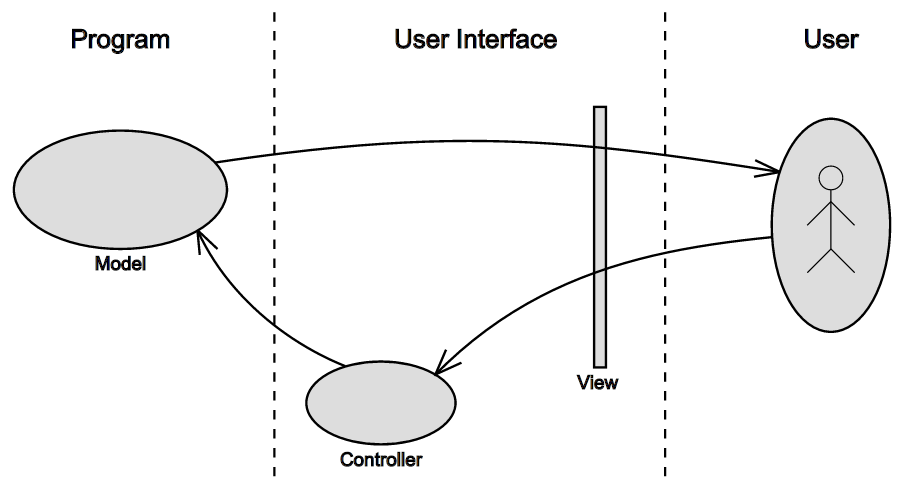
Доповнимо нашу діаграму місцем під юзкейси.



Ще користувачеві треба дати можливість смикати ці юзкейси і отримувати результат. Це можуть бути кнопки та інші графічні елементи введення-виведення, жести, розпізнавання і синтез мови. Будь-який варіант обміну даними і командами. вуаля:



Користувач смикає якийсь із юзкейсов, який, в свою чергу, проводить маніпуляції над програмою. Програма публікує результат або зміни в її стані. Все, що залишилося - це тільки роздати знайомі імена утворився компонентам

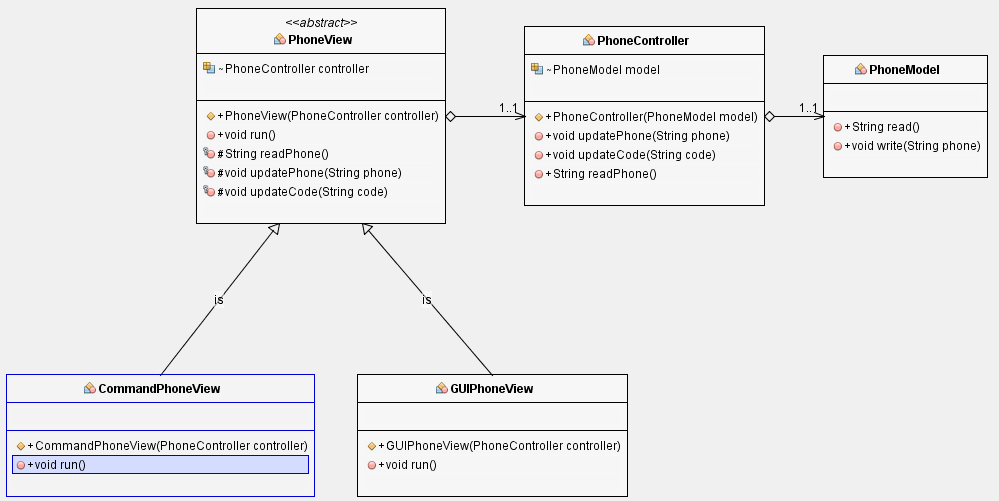


Коли модель публікує зміни, її не хвилює для кого, вона нічого не знає про View. Замість або разом зі View на тому кінці може бути інша підсистема.

Це був класичний варіант MVC - Active Model. Буває і так, що модель не сповіщає про зміни. Тоді цей обов'язок бере на себе контролер. Він знає, які маніпуляції виробляє над моделлю, і, очевидно, знає, які зміни в стані моделі можуть послідувати. Це Passive Model.

Розглянемно описану вище модель на прикладі.

Приклад 4. Реалізація задачі збереження телефоного номера із використанням MVC.



public class PhoneController {

PhoneModel model;

public PhoneController(PhoneModel model) {

this.model = model;

}

public void updatePhone(String phone) {

model.write(phone);

}

public void updateCode(String code) {

String phone = model.read();

// change operator code in phone

model.write(phone);

}

public String readPhone() {

return model.read();

}

}

class PhoneModel {

public String read() {

String phone = "";

// read phone from file

return phone;

}

public void write(String phone) {

// write phone to file

}

}

abstract class PhoneView {

PhoneController controller;

public PhoneView(PhoneController controller) {

this.controller = controller;

}

public abstract void run();

protected String readPhone() {

return controller.readPhone();

}

protected void updatePhone(String phone) {

controller.updatePhone(phone);

}

protected void updateCode(String code) {

controller.updateCode(code);

}

}

class CommandPhoneView extends PhoneView {

public CommandPhoneView(PhoneController controller) {

super(controller);

}

@Override

public void run() {

while (true) {

String command = "exit";

// 1. Ask to enter command (read, write or exit)

if (command == "read") {

String phone = readPhone();

// output to the console

} else if (command == "write") {

String phone = "";

//read phone from console

updatePhone(phone);

} else if (command == "change code") {

String phone = "";

//read code from console

updateCode(phone);

} else {

break;

}

}

}

}

class GUIPhoneView extends PhoneView {

public GUIPhoneView(PhoneController controller) {

super(controller);

}

@Override

public void run() {

// 1. Drow dialog window with two text boxes and two buttons

String phone = readPhone();

// 3. Show phone number in text box

// 4. Bound event onBtnClick with method updatePhone()

// 5. Bound event onBtnClick with method updateCode()

}

}

class Main {

public void run(String arg) {

PhoneModel model = new PhoneModel();

PhoneController controller = new PhoneController(model);

PhoneView view;

if (arg == "commandUI") {

view = new CommandPhoneView(controller);

} else {

view = new GUIPhoneView(controller);

}

view.run();

}

}

<https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D0%B0>

<https://tproger.ru/articles/mvc/>

http://www.mrwolf.ru/Kompyutery\_i\_soft/Pro4ee/11629