Одеський національний політехнічний університет

Інститут комп'ютерних систем

Кафедра інформаційних технологій

**КУРСОВА РОБОТА**

з дисципліни «Об'єктно-орієнтоване програмування»

Тема «Калькулятор для обчислення кількості необхідніх матеріалів для Виконання конкретної роботи в сфері будівництва, облаштування квартир новобудови»

Студента (ки) курсу групи

(Прізвище та ініціали)

Керівник РудніченкоМ.Д

(Посада, Вчене звання, науковий степень, прізвище та ініціали)

Національна шкала

Кількість балів: \_\_

Оцінка: ECTS \_\_\_\_ \_

члени КОМІСІЇ

(Підпис) (прізвище та ініціали)

(Підпис) (прізвище та ініціали)

(Підпис) (прізвище та ініціали)

м.Одеса - 2019 рік

Одеський національний політехнічний університет

Інститут комп'ютерних систем Кафедра інформаційних систем

**завдання**

НА курсова робота

(Прізвище, ім'я, по батькові) (група)

1. Тема роботи " Калькулятор для обчислення кількості необхідніх матеріалів для Виконання конкретної роботи в сфері будівництва, облаштування квартир новобудови "

1. Термін здачі студентом закінченої роботи
2. Початкові дані до проекту (роботи):

Зміст розрахунково-пояснювальної записки (ПЕРЕЛІК вопросам, Які Належить Розробити):вступ, основні понянія про мову java. Особливості плюси и мінусі, проектування, література

1. Перелік графічного матеріалу (з точним зазначеним обов'язкових креслень):

завдання бачено

Завдання Прийнято до Виконання

ЗМІСТ

[ВСТУП 4](#_Toc10727422)

[ГЛАВА 1 ОСНОВНІ Про МОВІ JAVA. ОСОБЛИВОСТІ, Плюси і мінуси 5](#_Toc10727423)

[1.1Основнi аспекти Java 5](#_Toc10727424)

[1.2Основніе знання про мови програмування 5](#_Toc10727425)

[1.3 Основні можливості Java 7](#_Toc10727426)

[1.4 Переваги і недоліки Java 8](#_Toc10727427)

[1.5 Що таке тестування і їх види 10](#_Toc10727428)

[1.6 Детальніше про види тестування 10](#_Toc10727429)

[1.7 Бібліотека J Unit 24](#_Toc10727430)

[1.8 TestNG 25](#_Toc10727431)

[2 ПРОЕКТУВАННЯ КАЛЬКУЛЯТОР ДЛЯ Обчислення КІЛЬКОСТІ НЕОБХІДНІХ МАТЕРІАЛІВ 28](#_Toc10727432)

[2.1 Backlog 28](#_Toc10727433)

[2.2 Розробка UML діаграм 28](#_Toc10727434)

[2.2.1 Use-Case 29](#_Toc10727435)

[2.2.2 Діаграма класів 31](#_Toc10727436)

[2.2.3 Діаграма послідовності 32](#_Toc10727437)

[2.3 Розробка форм інтерфейсу и описание процесса роботи з додатком 34](#_Toc10727438)

[ВИСНОВКИ 37](#_Toc10727439)

[ЛІТЕРАТУРА 38](#_Toc10727440)

ВСТУП

Потоки информации, что ціркулюють в мире, Який нас оточує, велічезні. Згідно смороду ма ють тенденцію до Збільшення. У будь-Якій компании, як великий, так и малий, вінікає проблема підрахунків будівельного матеріалу. В СУЧАСНИХ условиях однією з найбільш Поширення завдання, что вінікає при внутрішнього облицювання квартири, є розрахунок необхідного матеріалу. Особливо актуальною проблема организации завдання и справ вінікає в компаніях, де нужно ефективного распреділілісь роботу, пріскоріті, спростіті процес підбору матеріалу. Тому создания додатка, Який бі допомагать будівельнікам в таких условиях працювати найбільш ефективних Надзвичайно доцільнім.

Метою даної роботи є создания десктопного додатка, розрахованого на обчислення будівельного матеріалу під розміри кімнат.

Програмні Додатки можна реалізовуваті на різніх мовах програмування. Однією з таких є об'єктно-орієнтована мова програмування C #.

Мета роботи Полягає в розшіренні, закріпленні и сістематізації знань з дисципліни, что вівчається, Шляхом проектування і розробки програмного забезпечення, а самє десктопного додатка для обчислення кількості необхідніх матеріалів для Виконання конкретної роботи в сфері будівництва, облаштування квартир новобудови.

Додаток розробляється для персональних комп'ютерів під операційною системою Windows. Середовище розробки Visual Studio 2019, мова програмування C #.

ГЛАВА 1 ОСНОВНІ Про МОВІ JAVA. ОСОБЛИВОСТІ, Плюси і мінуси

1.1Основнi аспекти Java

Java- сильно типізований [об'єктно-орієнтована мова програмування](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), Розроблений компанією [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems) (В подальшому придбаної компанією [Oracle](https://ru.wikipedia.org/wiki/Oracle)). Програми Java зазвичай[транслюються](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80)в спеціальний[байт-код](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4), тому вони можуть працювати на будь-якої комп'ютерної архітектурі за допомогою[віртуальної Java-машини](https://ru.wikipedia.org/wiki/Java_Virtual_Machine). Дата офіційного випуску - 23 травня 1995 року. На 2018 рік Java - один з найпопулярніших мов програмування.

Спочатку мова називалася Oak ( «Дуб»), розроблявся Джеймсом Гослінгом для програмування побутових електронних пристроїв. Через те, що мова з такою назвою вже існував, незабаром Oak був перейменований в Java. Названий на честь марки кави Java, яка, в свою чергу, отримала найменування однойменного острова (Ява), тому на офіційній емблемі мови зображена чашка з гарячою кавою. Існує й інша версія походження назви мови, пов'язана з алюзією на каву-машину як приклад побутового пристрою, для програмування якого спочатку мова створювався. Відповідно до етимологією в російськомовній літературі з кінця двадцятого і до перших років ХХІ століття назва мови нерідко переводилося як Ява, а не транскрибувати, як це стало загальноприйнятим пізніше.

Згодом мова стала використовуватися для написання клієнтських додатків і серверного програмного забезпечення.

1.2Основніе знання про мови програмування

Програми на Java транслюються в байт-код Java, який виконується віртуальною машиною Java (JVM) - програмою, обробній байтовий код і передавальної інструкції обладнанню як інтерпретатор.

Перевагою подібного способу виконання програм є повна незалежність байт-коду від операційної системи і устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина. Іншою важливою особливістю технології Java є гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання. Часто до недоліків концепції віртуальної машини відносять зниження продуктивності. Ряд удосконалень кілька збільшив швидкість виконання програм на Java:

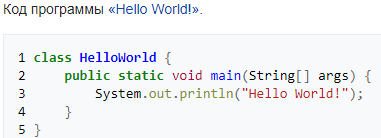
За даними сайту shootout.alioth.debian.org, для семи різних завдань час виконання на Java становить в середньому в півтора-два рази більше, ніж для C / C ++, в деяких випадках Java швидше, а в окремих випадках в 7 разів повільніше. З іншого боку, для більшості з них споживання пам'яті Java-машиною було в 10-30 разів більше, ніж програмою на C / C ++. Також примітно дослідження, проведене компанією Google, згідно з яким відзначається істотно нижча продуктивність і боольшой споживання пам'яті в тестових прикладах на Java в порівнянні з аналогічними програмами на C ++.

Ідеї, закладені в концепцію і різні реалізації середовища віртуальної машини Java, надихнули безліч ентузіастів на розширення переліку мов, які могли б бути використані для створення програм, що виконуються на віртуальній машині. Ці ідеї знайшли також вираз в специфікації загальномовна інфраструктури CLI, закладеної в основу платформи .NET компанією Microsoft.

1.3 Основні можливості Java

* [Автоматичне управління пам'яттю](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)).
* Розширені можливості обробки виняткових ситуацій.
* Багатий набір засобів фільтрації введення-виведення.
* Набір стандартних колекцій:[масив](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2),[список](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)),[стек](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA)і т.п.
* Наявність простих засобів створення мережевих додатків (у тому числі з використанням [протоколу](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) [RMI](https://ru.wikipedia.org/wiki/RMI)).
* Наявність класів, що дозволяють виконувати[HTTP](https://ru.wikipedia.org/wiki/HTTP)-запит і обробляти відповіді.
* Вбудовані в мову засоби створення багатопоточних додатків, які потім були перенести на багато мов (наприклад [Python](https://ru.wikipedia.org/wiki/Python)).
* Уніфікований доступ до[баз даних](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85):
* на рівні окремих[SQL](https://ru.wikipedia.org/wiki/SQL)-запит - на основі[JDBC](https://ru.wikipedia.org/wiki/JDBC),[SQLJ](https://ru.wikipedia.org/wiki/SQLJ);
* на рівні концепції об'єктів, що володіють здатністю до зберігання в базі даних - на основі [Java Data Objects](https://ru.wikipedia.org/w/index.php?title=Java_Data_Objects&action=edit&redlink=1)([англ.](https://en.wikipedia.org/wiki/Java_Data_Objects)) і [Java Persistence API](https://ru.wikipedia.org/wiki/Java_Persistence_API).
* Підтримка узагальнень (починаючи з версії 1.5).
* Підтримка лямбда, замикань, вбудовані можливості функціонального програмування (з 1.8).

Приклади програм на малюнку 1.1



Малюнок 1.1 - Приклад програми

1.4 Переваги і недоліки Java

переваги

Простота. Синтаксис мови був успадкований від C ++. Сьогодні на тлі Python, Groove або Go його важко назвати простим, проте тоді еволюційний вид дозволив привернути увагу Сі-разработчіков. Такая форма конструкцій не збільшує швидкість набору, однак читати, розуміти і відтворювати його просто.

Надійність. Надійність забезпечується двома принципами ООП. Ієрархія наслідування збільшує читаність коду і знижує кількість невимушених помилок. Сувора типізація. Розробнику доводиться виконувати більший обсяг роботи, але дані інтерпретується однозначно.

Крім того, спочатку в Java передбачався заборона прямого доступу до пам'яті, що також підвищувало б надійність. Але розробники залишили кілька лазівок, наприклад бекдор sun.misc.Unsafe, які ця заборона обходять.

Безпека. Крім збереження загальної форми конструкцій, Java в порівнянні з C ++ формально позбувся двох потенційних небезпек: покажчиків і множинного спадкоємства. На ділі обидві функції збережені, але представлені в іншому вигляді: замість покажчиків використовуються значення, а в множині спадкування беруть участь не класи, а інтерфейси. Проте, така особливість java програмування майже виключає можливий шкоди від неуважності розробника.

Зручність. Дослівно концепція Java звучить як: «Write once, run anywhere». Тобто виконуваність коду не залежить від операційної системи або встановленого ПО. Досягається це завдяки транслюванню в байт-код віртуальної машиною JVM.

А тут якраз і стала в нагоді ця особливість Java на android. Різноманітність виробників, моделей телефонів, характеристик - все це могло б негативно позначитися на роботі додатків, якби не існування такого універсального інструменту.

Продуктивність. Особливість Java, пов'язана з транслюванням в байт-код, позитивно позначається і на продуктивності кінцевих продуктів. За швидкістю виконання однотипні програми на Java поступаються в 1,5-2 рази програмами на C / C ++, при цьому перевершують JavaScript, Ruby, Python.

Розвинена екосистема. За 22 роки життя мова обріс десятками IDE і фреймворків, сотнями спільнот і форумів, тисячами бібліотек і плагінів. Все це сприятливо позначається на порозі входження в професію, затребуваності і якості вироблених за допомогою java продуктів.

Безумовно, кожен популярна мова програмування унікальний, кожен має свої недоліки і переваги. Особливості Java не мають революційного характеру, вони незначні, але в той же час фундаментальні. Саме те, що відрізняє хороший мову від кращого.

Закінчимо недоліки мови

Менш швидкий і більш вимогливий. Низьке, в порівнянні з іншими мовами, швидкодія, підвищені вимоги до обсягу оперативної пам'яті (ОП).

Великий обсяг стандартних бібліотек і технологій створює складності у вивченні мови. Постійний розвиток мови викликає наявність як застарілих, так і нових засобів, що мають одне і те ж функціональне призначення.

Вузьконаправлений. Швидше за нейтральне якість ніж недолік але це і робить його узконаправлениим мовою так як Java є повністю об'єктно-орієнтованою мовою. Наприклад, C ++ теж є об'єктно-орієнтованим, але в ньому є можливість писати програми не в об'єктно-орієнтованому стилі, а в Java так не можна.

Реалізовано з використанням інтерпретації Р-коду (байт-коду). Тобто програма спочатку транслюється в машінонезавісімий Р-код, а потім інтерпретується деякої програмою-інтерпретатором (віртуальна Java-машина, JVM).

1.5 Що таке тестування і їх види

тестування продуктивності

В[інженерії програмного забезпечення](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%B8%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F)-[тестування](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), Яке проводиться з метою визначення, як швидко працює обчислювальна система або її частина під певною[навантаженням](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B0_(%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D1%8F)). Також може служити для перевірки і підтвердження інших атрибутів якості системи, таких як[масштабованість](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C),[надійність](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B4%D1%91%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C)і споживання ресурсів. Тестування ділиться на 6 типів:

* Функціональне тестування (functional testing)
* Системне тестування (system testing)
* Тестування проіводітельності (performance testing)
* Регресійне тестування (regression testing)
* Модульне тестування (unit testing)
* Тестування безпеки (security testing)

1.6 Детальніше про види тестування

Функціональне тестування (functional testing)

Перевірка функціональності (тестування методом «чорного ящика») - перевірка відповідності програмного забезпечення вимогам, заявленим в специфікації. Може проводитися як повне тестування заявленої функціональність, так і перевірка тільки базової функціональності.

Тестування «білого ящика» (white box) Тестування на відповідність програмного продукту вимогам зі знанням внутрішньої структури реалізації системи (є в наявності вихідний код і технічні специфікації).

Тестування «чорного ящика» (black box) Тестування на відповідність програмного продукту вимогам без знання внутрішньої структури реалізації системи.

Тестування функціональності може проводитися в двох аспектах:

* вимоги
* Бізнес процеси

Тестування в перспективі «вимоги» використовує специфікацію функціональних вимог до системи як основу для дизайну тестових випадків (Test Cases). У цьому випадку необхідно зробити список того, що буде тестуватися, а що ні, пріоритезувати вимоги на основі ризиків (якщо це не зроблено в документі з вимогами), а на основі цього пріоритезувати тестові сценарії (test cases). Це дозволить сфокусуватися і не упустити при тестуванні найбільш важливий функціонал.

Тестування в перспективі «бізнес-процеси» використовує знання цих самих бізнес-процесів, які описують сценарії щоденного використання системи. У цій перспективі тестові сценарії (test scripts), як правило, ґрунтуються на випадках використання системи (use cases).

Переваги функціонального тестування:

* Імітує фактичне використання системи;
* Недоліки функціонального тестування:
* Можливість упущення логічних помилок в програмному забезпеченні;
* Імовірність надлишкового тестування.

Системне тестування (system testing)

Високорівнева перевірка функціоналу всієї програми або системи загалом.

Основним завданням системного тестування є перевірка як функціональних, так і не функціональних вимог в системі в цілому. При цьому виявляються дефекти, такі як неправильне використання ресурсів системи, непередбачені комбінації даних користувача рівня, несумісність з оточенням, непередбачені сценарії використання, відсутня або неправильна функціональність, незручність використання і т.д. Для мінімізації ризиків, пов'язаних з особливостями поведінки в системи в тому чи іншому середовищі, під час тестування рекомендуєтьсявикористовувати оточення максимально наближене до того, на яке буде встановлено продукт після видачі.

Можна виділити два підходи до системного тестування:

* На базі вимог (requirements based)
* Для кожного вимоги пишуться тестові випадки (test cases), перевіряючі виконання даної вимоги.
* На базі випадків використання (use case based)
* На основі уявлення про способи використання продукту створюються випадки використання системи (Use Cases). За конкретного випадку використання можна визначити один або більше сценаріїв. На перевірку кожного сценарію пишуться тест кейси (test cases), які повинні бути протестовані.

Тестування проіводітельності (performance testing)

Тестування, яке проводиться з метою визначення, як швидко працює система або її частина під певним навантаженням.

Тестування навантаження (load testing)

Тестування призначене для перевірки працездатності системи при стандартних навантаженнях і для визначення максимально можливого піку, при якому система працює правильно. На малюнок 1.2 візуально зображено тест продуктивності



Малюнок 1.2 - Візуальне зображення тесту продуктивності

Залежно від характеристик, які нам потрібно протестувати, тестування продуктивності ділиться на типи:

* Тестування навантаження (Loadtesting) - тестування часу відгуку програми на запити різних типів, з метою упевнитися, що додаток працює відповідно до вимог при звичайній користувальницької навантаженні.
* Стрес-тестування (Stresstesting) - тестування працездатності додатки при навантаженнях, що перевищують призначені для користувача в кілька разів. При стрес-тестуванні (найчастіше, тільки при ньому) ми можемо отримати реальні дані кордонів продуктивності додатка, досліджувати здатність програми обробляти виключення, її стабільність і стійкість. Саме в значно збільшеній навантаженні на додаток і полягає різниця між тестуванням продуктивності і стрес тестуванням.
* Тестування стабільності або напрацювання на відмову (Stability / Reliabilitytesting) досліджує працездатність додатки при тривалій роботі в часі, при нормальній для програми навантаженні.
* Об'ємне тестування (VolumeTesting) - тестування проводиться з збільшення не навантаження і часу роботи, а кількості використовуваних даних, які зберігаються і використовуються в додатку.
* Дуже часто при визначенні тестування продуктивності і його типах приходять до помилкового розуміння і плутанини даних термінів. Щоб уникнути цього і закріпити отримані знання, підведемо підсумок. Отже, тестування продуктивності - це перевірка таких функціональних вимог, як продуктивність і працездатність програми при різних навантаженнях і умовах. Залежно від досліджуваної характеристики програми, ми можемо виділити такі типи тестування як:
* Тестування навантаження (продуктивність при нормальних умовах).
* Стрес-тестування (працездатність, продуктивність і характеристики додатки при екстремальних навантаженнях).
* Тестування стабільності (при тривалій роботі).
* Об'ємне тестування (при збільшених обсягах оброблюваних даних).

Регресійне тестування (regression testing)

Регресійне тестування - це вид тестування спрямований на перевірку змін, зроблених в додатку або навколишньому середовищу (лагодження дефекту, злиття коду, міграція на іншу операційну систему, базу даних, веб сервер або сервер додатки), для підтвердження того факту, що існуюча раніше функціональність працює як і раніше ( см. також Санітарне тестування або перевірка узгодженості / справності). Регресійний можуть бути як функціональні, так і нефункціональні тести.

Як правило, для регресійного тестування використовуються тест кейси, написані на ранніх стадіях розробки і тестування. Це дає гарантію того, що зміни в новій версії програми не пошкодили вже існуючу функціональність. Рекомендується робити автоматизацію регресійних тестів, для прискорення подальшого процесу тестування і виявлення дефектів на ранніх стадіях розробки програмного забезпечення.

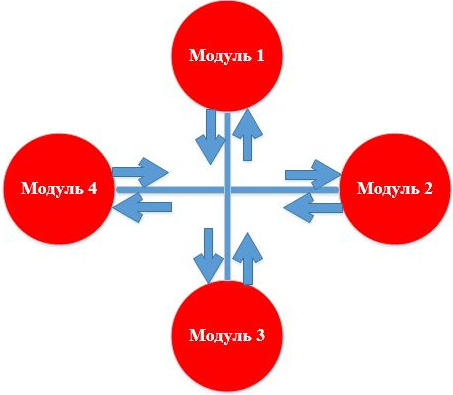
Сам по собі термін "Регресійне тестування", в залежності від контексту використання може мати різний зміст. СемКанер, наприклад, описав 3 основних типи регресивного тестування:

* Регресія багів (Bug regression) - спроба довести, що виправлена ​​помилка насправді не виправлена
* Регресія старих багів (Old bugs regression) - спроба довести, що нещодавня зміна коду або даних зламало виправлення старих помилок, тобто старі баги стали знову відтворюватися.
* Регресія побічного ефекту (Side effect regression) - спроба довести, що нещодавня зміна коду або даних зламало інші частини розробляється

Модульне тестування (unit testing)

Модульне тестування (Unit testing) - тестування кожної атомарної функціональності програми окремо, в штучно створеному середовищі. Саме потреба в створенні штучної робочого середовища для певного модуля, вимагає від тестувальника знань в автоматизації тестування програмного забезпечення, деяких навичок програмування. Дане середовище для деякого юніта створюється за допомогою драйверів і заглушок.

Кожна складна програмна система складається з окремих частин - модулів, що виконують ту або іншу функцію в складі системи. Для того, щоб упевнитися в коректній роботі всієї системи, необхідно спочатку протестувати кожен модуль системи окремо. У разі виникнення проблем при тестуванні системи в цілому це дозволяє простіше виявити модулі, що викликали проблему, і усунути відповідні дефекти в них. Таке тестування модулів окремо отримало назву модульного тестування. На малюнок 1.3 візуально зображено модульне тестування



Малюнок 1.3 - Візуально зображено модульне тестування

Драйвер - певний модуль тесту, який виконують тестований нами елемент.

Заглушка - частина програми, яка симулює обмін даними з тестованим компонентом, виконує імітацію робочої системи.

Заглушки потрібні для:

* Імітування відсутніх компонентів для роботи даного елемента.
* Подачі або повернення модулю певного значення, можливість надати тестеру самому ввести потрібне значення.
* Відтворення певних ситуацій (виключення або інші нестандартні умови роботи елемента).

Переваги модульного тестування.

Перш за все, потрібно окреслити рамки, в яких Юніт-тестування виправдано. По-перше, архітектура проекту повинна бути спроектована відповідно до ідей ООП (чіткий розподіл на класи, кожен з яких виконує свою певну функцію), що забезпечить систему грамотним розподілом на модулі. Також, модульне тестування повинно бути менш витратним при пошуку дефектів, ніж інші види тестів і має знижувати час налагодження коду.

переваги:

* Модульне тестування мотивує програмістів писати код максимально оптимізованим, проводити рефакторинг (спрощення коду програми, не зачіпаючи її функціональність), так як за допомогою Юніт-тестування можна легко перевірити працездатність даного компонента.
* Необхідність відділення реалізації від інтерфейсу (зважаючи на особливості модульного тестування), що дозволяє мінімізувати залежності в системі.
* Документація Юніт-тестів може служити прикладом «живого документа» для кожного класу, що тестується даними способом.
* Модульне тестування допомагає краще зрозуміти роль кожного класу на тлі всієї програмної системи.
* Також, при «розробці через тестування», яка активно використовується в екстремальному програмуванні, модульне тестування є одним з основних інструментів, що дозволяє розробляти модулі відповідно до вимог до даного модулю.
* Природно, отримавши реальний досвід роботи інженером якості, Ви зможете зовсім по-іншому охарактеризувати даний вид тестування. А поки, сподіваюся, ця стаття допоможе Вам підійти на крок ближче до можливості отримання реального досвіду.

Тестування безпеки (security testing)

Тестування безпеки - це стратегія тестування, яка використовується для перевірки безпеки системи, а також для аналізу ризиків, пов'язаних із забезпеченням цілісного підходу до захисту додатки, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних.

Принципи безпеки програмного забезпечення

Загальна стратегія безпеки грунтується на трьох основних принципах:

* Конфіденційність
* цілісність
* доступність

Конфіденційність - це приховування певних ресурсів або інформації. Під конфіденційністю можна розуміти обмеження доступу до ресурсу деякої категорії користувачів, або іншими словами, за яких умов користувач авторизований отримати доступ до цього ресурсу.

Цілісність. Існує два основних критерії при визначенні поняття цілісності:

* Довіра. Очікується, що ресурс буде змінений тільки відповідним способом певною групою користувачів.
* Пошкодження і відновлення. У разі коли дані пошкоджуються або неправильно змінюються авторизованим або авторизованим користувачем, ви повинні визначити на скільки важливою є процедура відновлення даних.

Доступність є вимоги про те, що ресурси повинні бути доступні авторизованому користувачеві, внутрішньому об'єкту або пристрою. Як правило, чим більш критичний ресурс тим вище рівень доступності повинен бути.

Види вразливостей. В даний час найбільш поширеними видами уразливості в безпеці програмного забезпечення є:

XSS (Cross-Site Scripting) - це вид уразливості програмного забезпечення (Web додатків), при якій, на генерованої сервером сторінці, виконуються шкідливі скрипти, з метою атаки клієнта.

XSRF / CSRF (Request Forgery) - це вид уразливості, що дозволяє використовувати недоліки HTTP протоколу, при цьому зловмисники працюють за такою схемою: посилання на шкідливий сайт установливается на сторінці, що користується довірою у користувача, при переході по шкідливої ​​посиланням виконується скрипт, який зберігає особисті дані користувача (паролі, платіжні дані і т.д.), або відправляє СПАМ повідомлення від особи користувача, або змінює доступ до облікового запису користувача, для отримання повного контролю над нею.

Code injections (SQL, PHP, ASP і т.д.) - це вид уразливості, при якому стає можливо здійснити запуск виконуваного коду з метою отримання доступу до системних ресурсів, несанкціонованого доступу до даних або виведення системи з ладу.

Server-Side Includes (SSI) Injection - це вид уразливості, що використовує вставку серверних команд в HTML код або запуск їх безпосередньо з сервера.

Authorization Bypass - це вид уразливості, при якому можливо отримати несанкціонований доступ до облікового запису або документам іншого користувача

Як тестувати ПО на безпеку?

Наведемо приклади тестування ПО на предмет уразливості в системі безпеки. Для цього Вам необхідно перевірити Ваше програмне забезпечення на наявності відомих видів вразливостей:

XSS (Cross-Site Scripting)

Самі по собі XSS атаки можуть бути дуже різноманітними. Зловмисники можуть спробувати вкрасти ваші куки, перенаправити вас на сайт, де відбудеться більш серйозна атака, завантажити в пам'ять будь-якої шкідливий об'єкт і т.д., всього навсього розмістивши шкідливий скрипт у вас на сайті. Як приклад, можна розглянути наступний скрипт, що виводить на екран ваші куки:

<Script> alert (document.cookie); </ script>

або скрипт робить редирект на заражену сторінку:

<Script> window.parent.location.href = 'http: // hacker\_site'; </ script>

або створює шкідливий об'єкт з вірусом і т.п .:

<Object type = "text / x-scriptlet" data = "http: // hacker\_site"> </ object>

Для перегляду більшої кількості прикладів рекомендуємо відвідати сторінку: XSS (Cross Site Scripting) ...

XSRF / CSRF (Request Forgery)

Найбільш частими CSRF атаками є атаки використовують HTML <IMG> тег або Javascript об'єкт image. Найчастіше атакуючий додає необхідний код в електронний лист або викладає на веб-сайт, таким чином, що при завантаженні сторінки здійснюється запит, виконує шкідливий код. приклади:

IMG SRC

<img src = "http: // hacker\_site /? command">

SCRIPT SRC

<Script src = "http: // hacker\_site /? Command">

Javascript об'єкт Image

<script>

var foo = new Image ();

foo.src = "http: // hacker\_site /? command";

</ Script>

Code injections (SQL, PHP, ASP і т.д.)

Вставки виконуваного коду розглянемо на прикладі коду SQL.

Форма входу в систему має 2 поля - ім'я та пароль. Обробка відбувається в базі даних через виконання SQL запиту:

SELECT Username

FROM Users

WHERE Name = 'tester'

AND Password = 'testpass';

Вводимо коректне ім'я 'tester', а в поле пароль вводимо рядок:

testpass 'OR' 1 '=' 1

У підсумку, Якщо поле не має відповідних валідацій або обробників даних, може розкритися вразливість, що дозволяє зайти в захищену паролем систему, т.к.SQL запит прийме наступний вигляд:

SELECT Username

FROM Users

WHERE Name = 'tester'

AND Password = 'testpass 'OR' 1 '=' 1 ';

Умова '1' = '1' завжди буде істинним і тому SQL запит завжди буде повертати багато значень.

Server-Side Includes (SSI) Injection

Залежно від типу операційної системи команди можуть бути різними, як приклад розглянемо команду, яка виводить на екран список файлів в OS Linux:

<! - # exec cmd = "ls" ->

Authorization Bypass. Користувач А може отримати доступ до документів користувача Б. Припустимо, є реалізація, де при перегляді свого профілю, що містить конфіденційної інформацію, в URL сторінки передається userID, а даному випадку є сенс спробувати підставити замість свого userID номер іншого користувача. І якщо ви побачите його дані, значить ви знайшли дефект.

Тестування локалізації (localization testing)

Тестування локалізації - це процес тестування локалізованої версії програмного продукту. Тестування локалізації дозволяє перевірити наскільки добре адаптований продукт для певної цільової аудиторії відповідно до її культурними особливостями. Як правило, тут розглядається культурний і мовний аспекти, зокрема переклад користувацького інтерфейсу і відповідної документації і файлів на іншу мову, а також формати валют, чисел, часу і телефонних номерів, і ін.

Наші висококваліфіковані фахівці займаються локалізацією російської, англійської, німецької, французької та італійської та інших версій продуктів.

В нашій команді працюють професійні лінгвісти та QA інженери, які спільно працюють над виявленням помилок локалізації.

У тестування локалізації входить перевірка правильності перекладеного контенту, різних елементів інтерфейсу, помилок і системних повідомлень, перевірка розділу FAQ / Help і допоміжної документації.

Мета проведеного тестування - перевірити багатомовний інтерфейс програми або сайту на наявність помилок перекладу, правильності поштових адрес, імені та прізвища, валют, формату дати і часу, і ін.

Як правило, ми не тестуємо на наявність орфографічних і граматичних помилок. Чекліст тестування локалізації включає в себе:

* Визначення того, які мови підтримуються додатком;
* Перевірка правильності перекладу відповідно до тематики програми;
* Перевірка правильності перекладу елементів інтерфейсу програми;
* Перевірка перекладу розділу FAQ / Help і допоміжної документації.
* Тестування інтернаціоналізації є проектування і створення продукту і документації, використовуючи техніки, спрощують локалізацію додатки. Сюди входять:
* створення продукту з урахуванням стандарту кодування Unicode, який підтримує практично всі мови світу;
* створення можливості підтримки елементів програми, які неможливо локалізувати стандартним чином (наприклад, вертикально розташовані ієрогліфи або написання арабських пропозицій справа наліво);
* створення продукту з можливістю підтримки нових локалей при бажанні користувача, що дозволяє уникнути труднощів при інтеграції продукту для країн з іншою культурою.

Юзабіліті тестування (usability testing)

Юзабіліті-тестування (Usability - зручність) - це перевірка програмного продукту на відповідність до вимог в плані зручності використання програми. Таким чином, за допомогою юзабіліті-тестування ми можемо визначити ергономічність (пристосованість до використання) програми. Тестування зручності користування - це метод тестування, спрямований на встановлення ступеня зручності використання, навченості, зрозумілості і привабливості для користувачів продукту, що розробляється в контексті заданих умов. Виявляти проблеми, пов'язані зі специфічним механізмом інтерфейсу визначати, чи існують проблеми з зручністю інтерфейсу для навігації, використання основного функціоналу.

Перевірка юзабіліті додатки полягає в:

* Оцінка відповідності дизайну додатки до його функціональності, заданої замовником.
* Аналіз використовуваних графічних елементів, колірного оформлення з точки зору сприйняття.
* Оцінці зручності навігації і посилальної структурі.
* Аналіз текстового наповнення сайту.
* Оцінка зручності використання функціями програми (сервісами, якщо це сайт).
* Аналіз шрифтового оформлення тексту.



Малюнок для 1.4 - Візуалізація юзабіліті тестування

10 правил проектування призначеного для користувача інтерфейсу, Norman

1. Інформативність системи - користувач завжди повинен знати поточний статус програми.
2. Наближеність додатки до реального світу - діалог з користувачем повинен вестися на зрозумілому йому мові, остерігаючись використання незрозумілою термінології.
3. Система повинна мати виходи - додаток завжди повинно мати «запасні виходи» з будь-якої функціональності, які користувач помилково запустив.
4. Однозначність. Всі терміни, функції і поняття повинні описуватися в єдиному тлумаченні - у користувача не повинно виникнути плутанини.
5. Передбачливість. Система повинна всіляко «оберігати» користувача від можливих помилок.
6. Наочність. Користувач не повинен ламати голову в спробах зрозуміти, що йому потрібно робити або намагаючись пригадати, як він досяг того чи іншого стану системи. Можливі маніпуляції з програмою повинні бути постійно наочними.
7. Гнучкість і ефективність. Надавайте досвідченим користувачам можливість уникати рутинних дій, і в той же самий час, необхідно приховувати розширення функціоналу від недосвідчених.
8. Лаконічність і точність. Діалоги повинні містити тільки ту інформацію, яку необхідно донести до користувача, нічого зайвого.
9. Лояльність до помилок. Інформація про помилки повинна бути зрозумілою і містити підказки до подальших дій.
10. Постійна довідка. Як би інформативно була спроектована система - вона завжди повинна містити розділ довідки та документації.

1.7 Бібліотека J Unit

Unit testing (юніт тестування або модульне тестування) - полягає в ізольованій перевірці кожного окремого елемента шляхом запуску тестів в штучному середовищі. Для цього необхідно використовувати драйвери і заглушки. Поелементне тестування - найперша можливість реалізувати вихідний код. Оцінюючи кожен елемент ізольовано і підтверджуючи коректність його роботи, точно встановити проблему значно простіше ніж, якби елемент був частиною системи. Unit (Елемент) - найменший компонент, який можна скомпілювати.

Фреймворк jUnit є досить вдалим рішенням завдань, пов'язаних з тестуванням java додатків. Зростаюча популярність призвела до створення подібних фреймворків для інших мов. Ось деякі з них:

1. Для С ++ була реалізована CPP Unit;
2. JavaScript може використовуватися спільно з JSUnit;
3. Для C # розробники створили NUnit;
4. Perl скрипти можна тестувати за допомогою Test :: Unit;
5. PHP код можуть тестувати за допомогою PHP Unit модуля.

jUnit застосовується для модульного тестування, яке дозволяє перевіряти на правильність окремі модулі вихідного коду програми. Перевага даного підходу полягає в ізолювання окремо взятого модуля від інших. При цьому, мета такого методу дозволяє програмісту упевнитися, що модуль, сам по собі, здатний працювати коректно. jUnit вдає із себе бібліотеку класів. Для демонстрації основних можливостей цієї бібліотеки, можна написати примітивний клас:

public class Salary {

private int value;

private String type;

public Salary (int v, String t) {

value = v;

type = t;

}

public Salary add (Salary s) {

return new Salary (value + s.getValue (), type);

}

public int getValue () {

return value;

}

}

1.8 TestNG

Як пояснення наведу приклад роботи:

Найпростіший тест. До того, як ми почнемо, давайте подумаємо над тим, а що ж власне ми будемо перевіряти? Пропонується далеко не ходити і написати невеликий клас, який буде в конструкторі отримувати 2 числа і буде мати метод getResult (), який повертає нам твір цих 2-х чисел. Просто і зі смаком.

Код буде виглядати приблизно ось так:

public class Calculator {  
 private int first;  
 private int second;  
   
 public Calculator(int first, int second) {  
 this.first = first;  
 this.second = second;  
 }  
  
 public int getResult(){  
 return first \* second;  
 }  
}

Покладемо його туди, де за правилами maven'а повинні лежати вихідні, над якими виробляються тести - папку main / java. Зрозуміло, це правило не строге, так як тести можуть проводитися над будь-яким стороннім модулем, що знаходяться в принципі в іншому проекті. Але якщо ми пишемо тести на роботу локально створеного класу, то і перебувати він повинен в певному місці.

Так як Ваш проект був налаштований на використання maven, то Вам доведеться слідувати деяким правилами структуризації коду, встановленими maven'ом. За нею, все класи з тестами повинні бути в папці src / test / java. Створіть її, якщо вона не створена. Всередині неї давайте створимо пакет, в якому будуть зберігатися класи з тестами. Усередині пакета - давайте створимо найпростіший клас тесту. Назвемо його, скажімо, ОurCoolTest. На відміну від JUnit - у TestNG немає строгого правила з приводу назви класу. Але все таки краще класи з тестами називати так, щоб було зрозуміло, для чого вони.

У підсумку після всього вищеописаного, структура вашого проекту буде виглядати приблизно ось так:

my-app  
|-- pom.xml  
`-- src  
 `-- main  
 `-- java  
 `-- com  
 `-- perfecttest  
 `-- somepackage  
 `-- Calculator.java  
 `-- test  
 `-- java  
 `-- com  
 `-- perfecttest  
 `-- somepackage  
 `-- OurCoolTest.java

2 ПРОЕКТУВАННЯ КАЛЬКУЛЯТОР ДЛЯ Обчислення КІЛЬКОСТІ НЕОБХІДНІХ МАТЕРІАЛІВ

2.1 Backlog

Бачення системи:система представляет собою десктопних програм, что дозволяє корістувачеві вводіті дані квартири в Якій хоче сделать ремонт.

Прізначені для користувача роли:користувач, адміністратор.

Робота з одними історіямі:

Розробляємо історії за принципом Як <користувач>, я можу <дію>, для того, щоб <мета>

1. Як користувач я можу використовуват програму для Власний потреб.
2. Як адміністратор я можу вносіті Зміни в базу Даних Різні будматеріали.
3. Як користувач я можу вводіті свои Особисті дані, щоб завести обліковій запис.

* Повінні буті введені справжні дані користувача.
* Обліковій Запис повинен буті унікальною.

1. Як користувач я можу Изменить дані облікового запису.
2. Як користувач я можу видалити свой обліковій запис.

* Для відалення та патенти підтвердження адміністратора.

1. Як користувач я можу ввести метраж свого приміщення, для того, щоб отріматі необхідні, раціональні варіації матеріалів для ремонту.
2. Як користувач я можу зберігаті результати свого списку матеріалів для ремонту приміщень.

2.2 Розробка UML діаграм

UML ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) Unified Modeling Language - уніфікована мова моделювання) - мова [графічного](https://ru.wikipedia.org/wiki/Визуализация) Опису для [об'єктного моделювання](https://ru.wikipedia.org/w/index.php?title=Объектное_моделирование&action=edit&redlink=1) в області [розробки програмного забезпечення](https://ru.wikipedia.org/wiki/Разработка_программного_обеспечения), для [моделювання](https://ru.wikipedia.org/wiki/Моделирование) [бізнес-процесів](https://ru.wikipedia.org/wiki/Бизнес-процесс), [системного проектування](https://ru.wikipedia.org/wiki/Системное_проектирование) и відображення [організаційніх структур](https://ru.wikipedia.org/wiki/Организационная_структура).

UML є мовою широкого профілю, це -[Відкритий стандарт](https://ru.wikipedia.org/wiki/Открытый_стандарт), Який вікорістовує графічні Позначення для создания [абстрактної моделі](https://ru.wikipedia.org/wiki/Абстрактная_модель) [системи](https://ru.wikipedia.org/wiki/Система), Званої UML-моделлю. UML БУВ Створений для визначення, візуалізації, проектування та Документування, в основному, програмних систем. UML НЕ є мовою програмування, но на підставі UML-моделей можлива[генерація коду](https://ru.wikipedia.org/wiki/Кодогенерация).

2.2.1 Use-Case

Діаграмі варіантів использование (use-case) опісують функціональне призначення системи або ті, что система винна делать.

Розробка діаграмі переслідує Такі цілі:

* візначіті Спільні кордони и контекст модельованої предметної області;
* сформулюваті Загальні вимоги до функціонального поведінкі проектованої системи;
* Розробити віхідну концептуальну модель системи для ее подальшої деталізації у форме логічніх и фізичних моделей;
* підготуваті віхідну документацію для взаємодії розробніків системи з ее замовниками и Користувачами.

Суть діаграмі варіантів использование Полягає в проектованої системе Подання у виде безлічі сутности або акторів, что взаємодіють з системою за помощью варіантів использование. При цьом актором (actor) або дійовою особою назівається будь-яка сутність, что Взаємодіє з системою ззовні. Це может буті людина, технічний Пристрій, програма або будь-яка Інша система, яка может служити Джерелом впліву на моделируемую систему так, як Визначіть сам розробник. Варіант использование служити для Опису сервісів, Які система надає актору. Діаграма варіантів использование может доповнюватіся Пояснювальна текстом, Який розкріває Зміст або семантику складових ее компонентів.

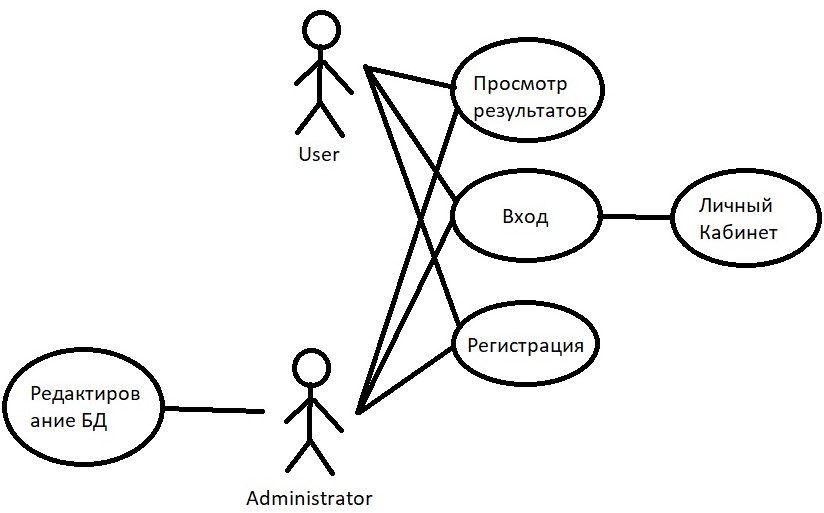
варіант использование

Окремий варіант использование позначається на діаграмі еліпсом, усередіні которого містіться его коротка назва або имя у форме дієслова з Пояснювальна словами.

Мета варіанту использование Полягає в тому, щоб візначіті закінчений аспект або фрагмент поведінкі деякої сутності без Розкриття ее внутрішньої структури. В якості такой сутності может віступаті система або будь-який елемент моделі, Який володіє, власною поведінкою.

КОЖЕН варіант использование відповідає окремий сервісу, Який надає моделируемая Сутність За запитом актора, тобто візначає способ! Застосування цієї сутності. Сервіс, Який ініціалізується за Запитів актора, є закінченою неподільну послідовність Дій. Це означає, что после того як система закінчіть обробка запиту, вона винна вернуться в початковий стан, щоб буті готовою до Виконання Наступний Запитів.

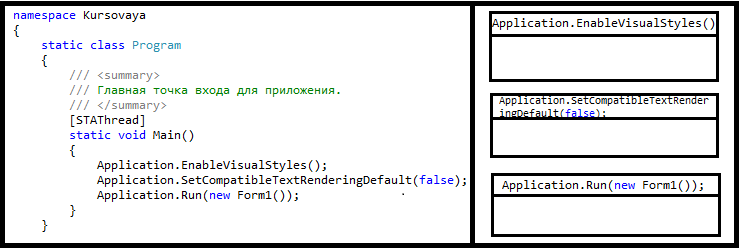
Варіанти использование могут застосовуватіся як для Специфікації зовнішніх вимог до проектованої системи, так и для Специфікації функціонального поведінкі Вже існуючої системи. Безліч варіантів использование в цілому винне візначаті всі Можливі Сторони очікуваної поведінкі системи. Кроме цього, варіанти использование неявно встановлюються вимоги, что визначаються, як актори повінні взаємодіяті з системою, щоб мати можлівість коректно працювати з надання сервісамі. Діаграма варіантів использование представлена ​​на малюнку 2.1



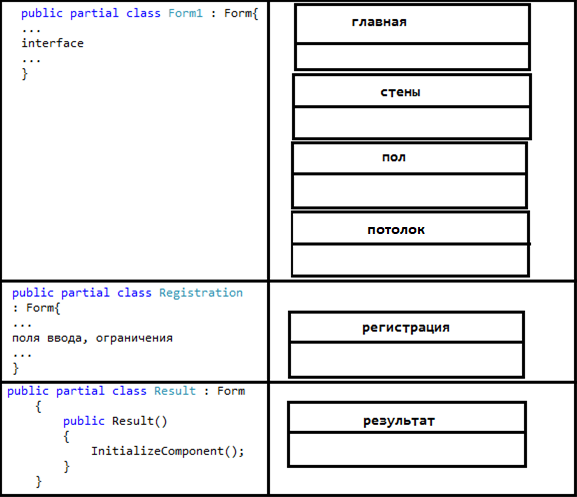
Малюнок 2.1 - Електронна Діаграма варіантів использование

2.2.2 Діаграма класів

Діаграма класів ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) Static Structure diagram) - структурна [Діаграма](https://ru.wikipedia.org/wiki/Диаграмма_(UML)) мови моделювання [UML](https://ru.wikipedia.org/wiki/UML), Що демонструє Загальну структуру ієрархії [класів](https://ru.wikipedia.org/wiki/Класс_(программирование)) системи, їх кооперацій, [атрібутів](https://ru.wikipedia.org/wiki/Поле_класса) (Полів), [методів](https://ru.wikipedia.org/wiki/Метод_(языки_программирования)), Інтерфейсів и взаємозв'язків между ними. Широко застосовується НЕ только для Документування та візуалізації, но кож для конструювання с помощью прямого або зворотнього проектування. Діаграма класів представлені на малюнку 2.2 и 2.3.



Малюнок 2.2 - головний клас від которого все почінається

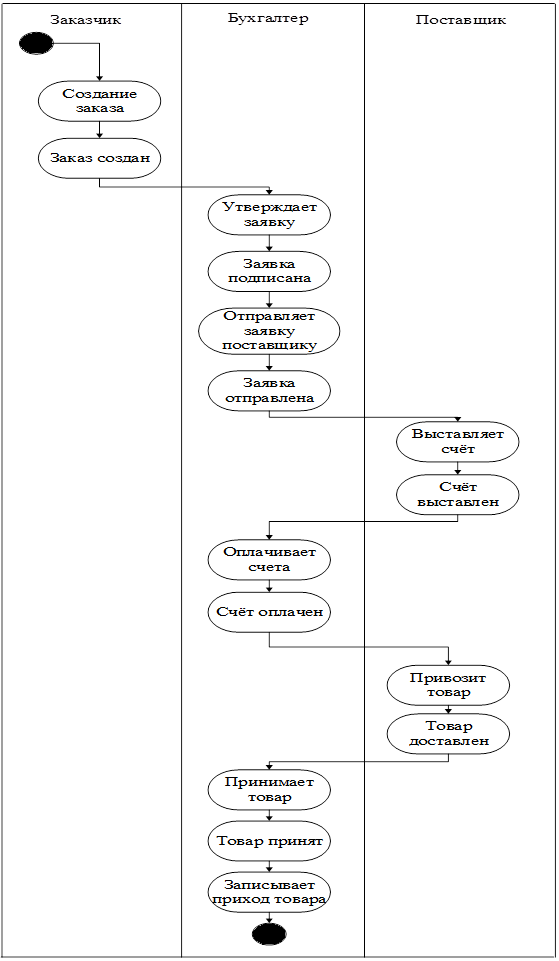


Малюнок 2.3 - класи форм інтерфейсу, реєстрації и результату.

2.2.3 Діаграма послідовності

Діаграма послідовності ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) sequence diagram) - [Діаграма](https://ru.wikipedia.org/wiki/Диаграмма), На Якій для Деяк набору об'єктів на єдіній тімчасової осі показань життєвий цикл будь-которого Певного об'єкта (создания-діяльність-знищення якоїсь сутності) и Взаємодія акторів (дійовіх осіб) ІС в рамках якої-небудь певної прецеденту (відправки Запитів и Отримання Відповідей). Вікорістовується в мові[UML](https://ru.wikipedia.org/wiki/UML).

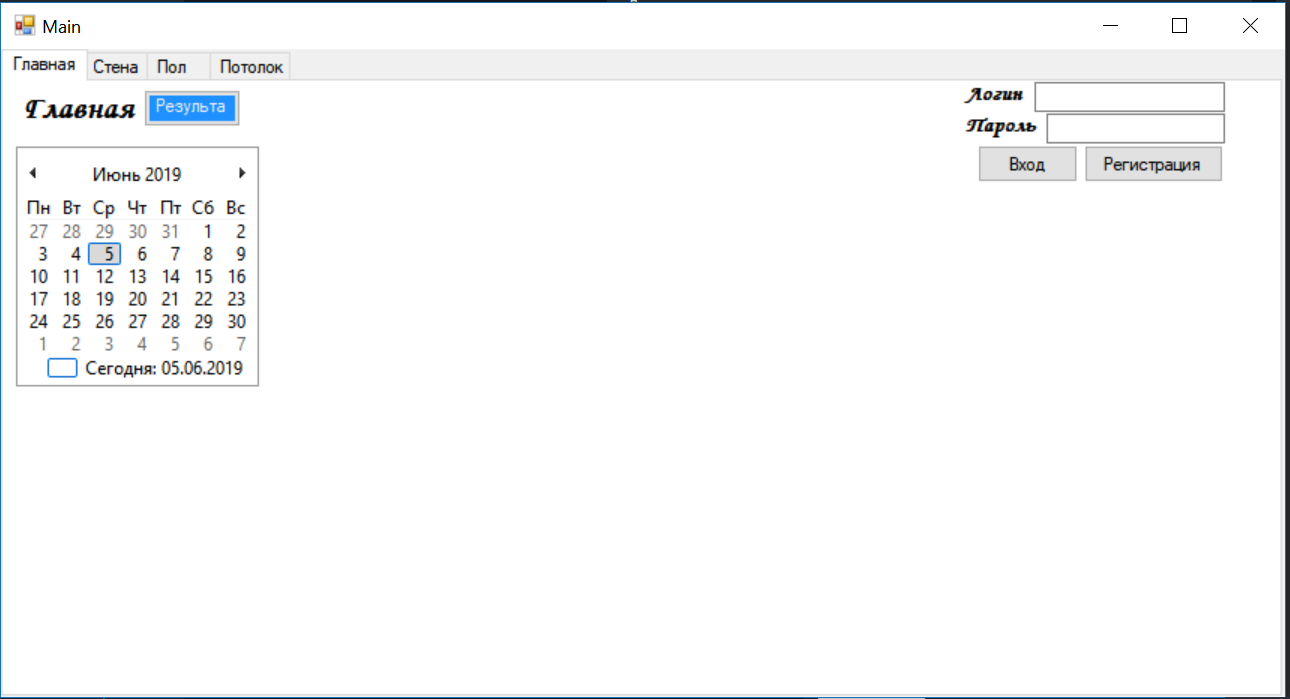
Основними елементами діаграмі послідовності є Позначення[об'єктів](https://ru.wikipedia.org/wiki/Объект_(программирование)) (Прямокутник з назв об'єктів), вертикальні «Лінії життя» ([англ.](https://ru.wikipedia.org/wiki/Английский_язык)lifeline), что відображають плин часу, прямокутник, что відображають діяльність об'єкта або Виконання ним певної Функції (прямокутник на пунктирною «Лінії життя»), и стрілки, что показують обмін сигналами або повідомленнями между об'єктами. Діаграма послідовності представлені на малюнку 2.4



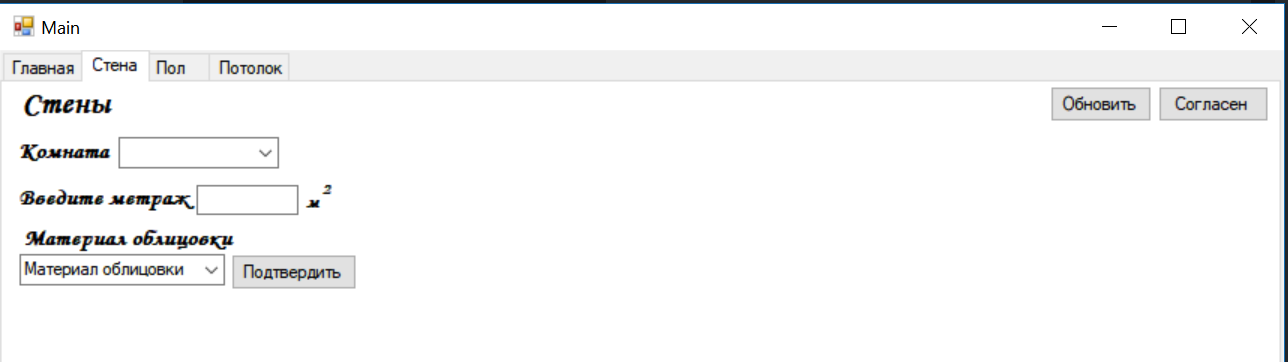
Малюнок 2.4 - Діаграма послідовності

2.3 Розробка форм інтерфейсу и описание процесса роботи з додатком

На малюнку 2.5 наведено скрін головній странице де доступна реєстрація, перехід на розрахункові форми де и відбуватіметься обчислення.

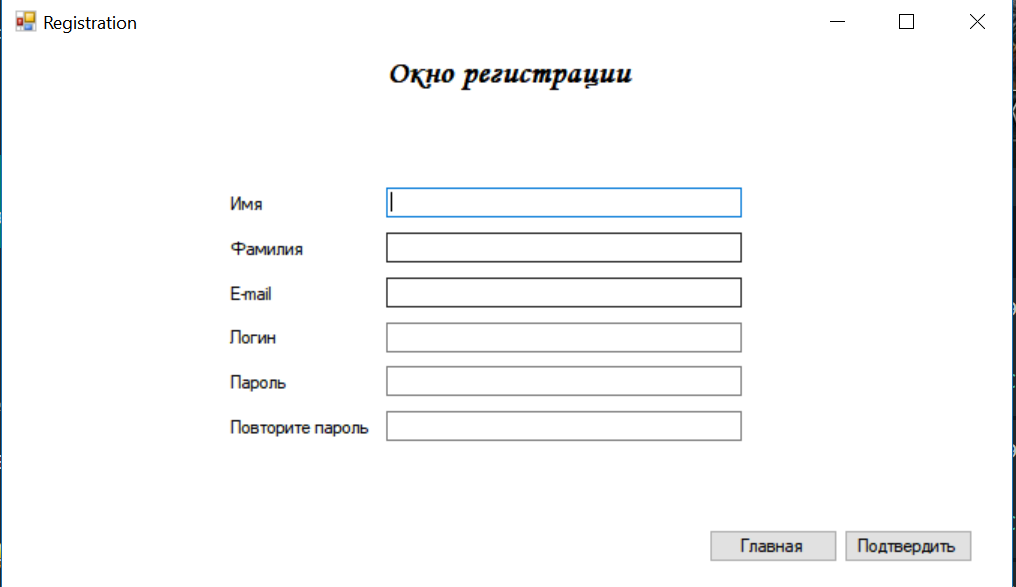


Малюнок 2.5 - головна форма

При натісканні на вкладку «Стіні» можна Побачити Малюнок 2.6 на якому Доступні для введення в осередку Даних, и с помощью Введення Даних буду проводитись розрахунки калькудятора, за таким принципом були створенні и інші 2 форми. 

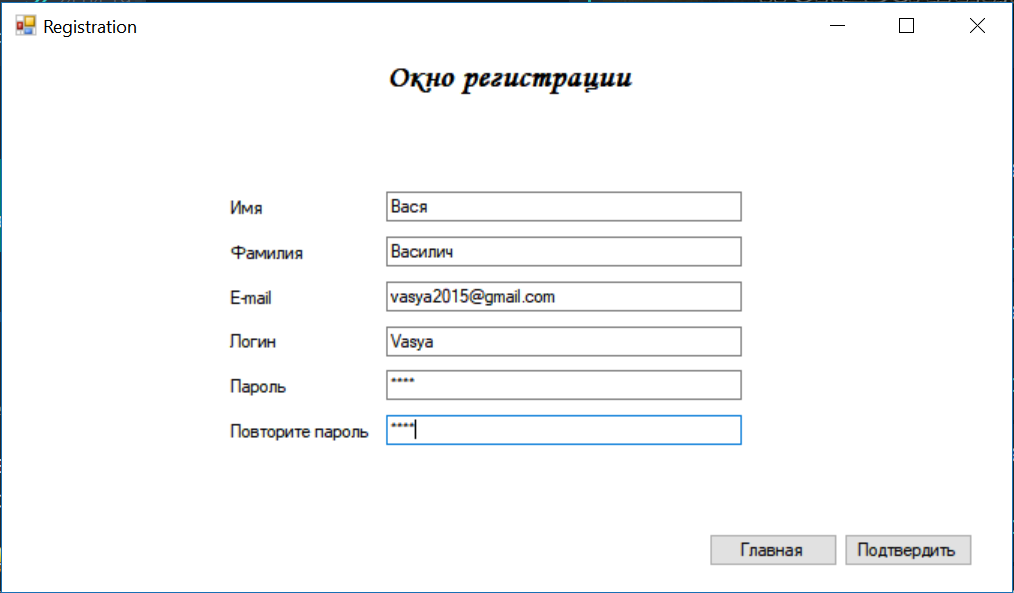
Малюнок 2.6 - вкладка Обчислення «стіні»

Для входу в аккаунт нужно зареєструватіся. Реєстрація проиходит у вкладка «Реєстрація» де задані обмеження по введенню. Паролі обов'язково повінні збігатіся Малюнок 2.7



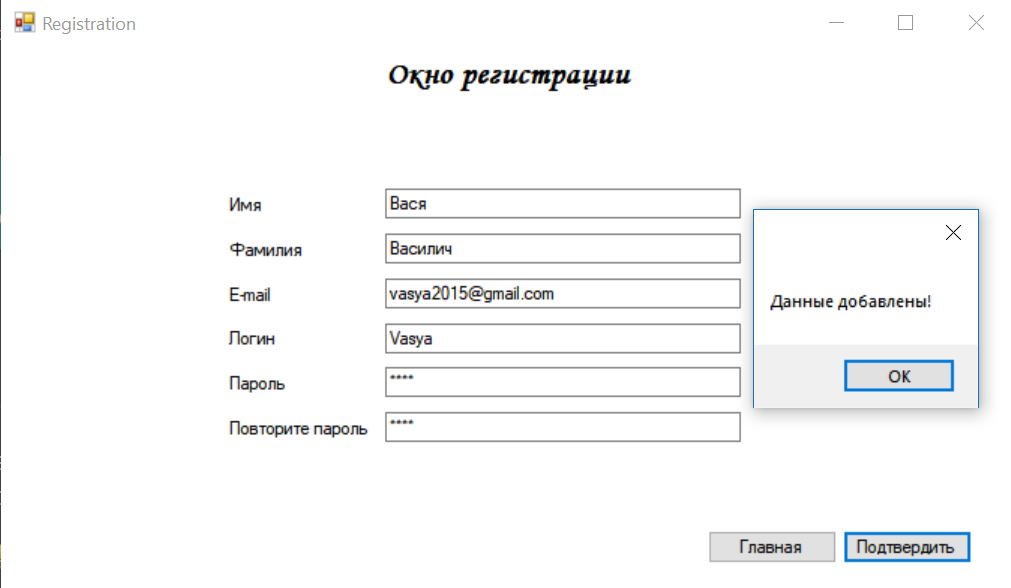
Малюнок 2.7 - форма реєстрації

Нижчих наведено приклад Введення Даних и Заповнення всех полів на Малюнок 2.8



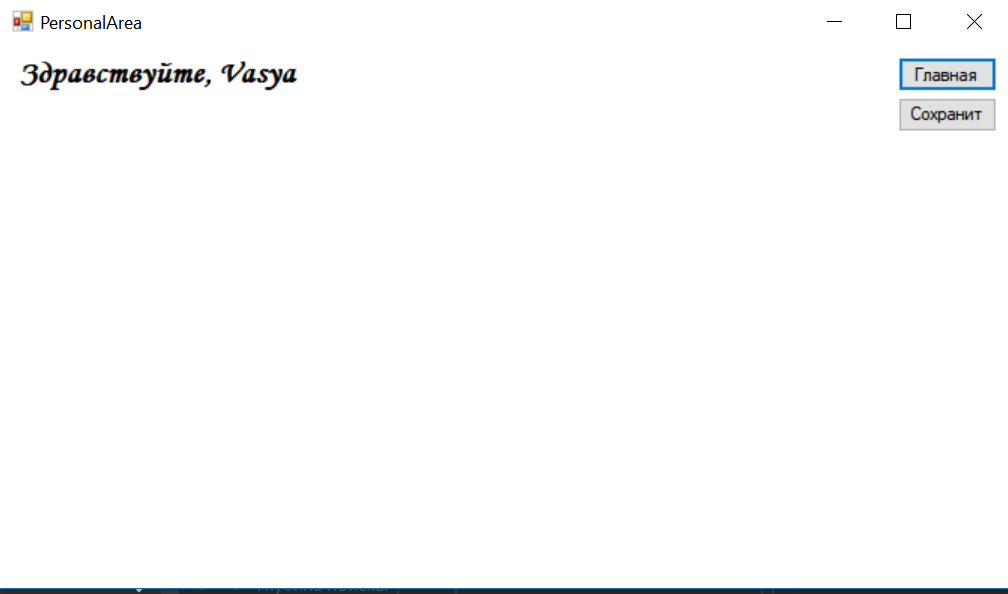
Малюнок 2.8 - реєстрація з заповненості полями

Користувач БУВ Створення і внесень в базу Даних, цього свідетільствует ібораженіе на Малюнок 2.9



Малюнок 2.9 - користувач зареєстрованій

Ввели данні и увійшлі в Профіль Васі, де могут зберігатіся его закасі Малюнок 2.10



Малюнок 2.10 - Особистий кабінет

ВИСНОВКИ

На сьогоднішній день Програмні Застосункі дозволяють ефективного зберігаті, структурувати і сістематізуваті великі обсяги даних. І без них неможливо уявіті роботу більшості промислових, торгівельних організацій.

Використання технологій дозволяє зберегти значні кошти, а головне и година для отримання необхідної інформації. Виконання даної курсової роботи дозволило мені:

* розшіріті, закріпіті і сістематізуваті теоретичні та практичні знання в рамках дисципліни, що вівчається, стосовно спеціфікі розробки сучасного програмного забезпечення;
* розвінутості навички ведення командних теоретичного і практичних досліджень в напрямку проектування програмного забезпечення;
* набути досвіду обробки, аналізу та сістематізації результатів практичних досліджень;
* сформувати навички роботи з науковою літературою та правильного оформлення дослідніцькіх робіт;
* закріпіті навички, науково оформляти і викладати свої думки, висновки і результати дослідження.

В ході виконання першого розділу був проведення аналіз розробки застосунку для персонального комп'ютера. Була представлена ​​структура застосунку. Проведено аналіз аналогів на ринку. В ході виконання було спроектована архітектура застосунку. Було описано бачення системи, корістувацькі ролі, корістувацькі історії, сформованому backlog продукту. Розроблені прототипи графічного інтерфейсу, віявлені. Були напісані діаграмі класів та послідовностей. Описаний процес роботи програмного застосунку. Наведені скріншоти інтерфейсу. Я отримав досвід роботи з мовою програмування C#, а також практичні навички розробки десктопного застосунку. Я переконався у тому, що отриманий досвід безперечно знадобиться мені у професії.

ЛІТЕРАТУРА

1. <https://ru.wikipedia.org/wiki/Java>
2. https://www.syl.ru/article/199389/new\_sovremennyiy-yazyik-programmirovaniya-vyisokogo-urovnya-primeryi-i-sravneniya
3. https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%B5%D1%80%D1%85%D0%B2%D1%8B%D1%81%D0%BE%D0 % BA% D0% BE% D1% 83% D1% 80% D0% BE% D0% B2% D0% BD% D0% B5% D0% B2% D1% 8B% D0% B9\_% D1% 8F% D0% B7 % D1% 8B% D0% BA\_% D0% BF% D1% 80% D0% BE% D0% B3% D1% 80% D0% B0% D0% BC% D0% BC% D0% B8% D1% 80% D0 % BE% D0% B2% D0% B0% D0% BD% D0% B8% D1% 8F
4. http://fb.ru/article/168378/spisok-yazyikov-programmirovaniya-yazyiki-programmirovaniya-nizkogo-i-vyisokogo-urovnya
5. https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA\_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0 % B0% D0% BC% D0% BC% D0% B8% D1% 80% D0% BE% D0% B2% D0% B0% D0% BD% D0% B8% D1% 8F
6. https://studopedia.su/7\_36667\_yaziki-programmirovaniya-nizkogo-urovnya.html
7. http://maksakov-sa.ru/ProgrProd/YazProgr/index.html
8. Початок роботи з Java SE Шевченко Д.В. Проскуріна Г.
9. Java: Методи програмування І.М. Блінов В.С. романчик
10. Басюк Т. М. Основи інформаційних технологій: навч. посібник / Т.М. Басюк, Н.О. Думанський, О.В. Пасічник; за наук. ред. В.В. Пасічніка.- Л .: Новий Світ-2000, 2011.- 390 с.
11. Васильєв О.М. Об'єктно-орієнтоване програмування. Java; Навчальний посібник - СПб.: Пітер, 2011. - 400с.
12. Вірт Н. Алгоритми и Структури Даних: Пер. з англ. / Н. Вірт. - СПб.: Невський діалект, 2008. - 352 с.
13. Глава М.Г. Організація баз Даних та знань: Конспект лекцій. [Електронний ресурс] - Режим доступу: http://library.opu.ua.
14. Гіберт Ш.Б. Java. Повне керівництво Опис, 8-е видання; Посібник програміста. - СПб.: Київ, 2013. - 175с.
15. Копитко М.Ф., Іванків К.С. Основи програмування мовою Java: Тексти лекцій. - Львів: Видавничий центр ЛНУ ім. Івана Франка, 2002. - 83 с.
16. Фаулер Мартін. UML. Основи. Третє видання. Короткий посібник по стандартній мові об'єктного моделювання. - Символ Плюс, 2006. - 501 с.
17. Шілдт Г., Холмс Д. Мистецтво програмування на Java. - М.: Видавничий дім "Вільямс", 2005. - 336 стор.
18. <https://metanit.com/sharp/tutorial/>
19. Албахарі Д., Албахарі Б.C # 6.0. Довідник. Повний опис мови. 6-е изд. Київ 2016 - 1040
20. Подбельский В.В. Мова C #. Базовий курс. 2-е изд.