УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Кафедра інформаційних технологій та програмної інженерії

КУРСОВА РОБОТА

з Об’єктно-орієнтованого програмування

на тему: «**Класовий тип для роботи з структурами типу «Вектор»**

Студента II курсу ІПЗс-2019 групи

Спеціальності 121 «Інженерія програмного забезпечення»

Ісарук В.В.

Керівник _____к.т.н., Пашкевич О.П.

Національна шкала _____

Кількість балів _____ Оцінка: ECTS _____

м. Івано-Франківськ

2021

_____Університет Короля Данила_____

назва вищого навчального закладу

Кафедра ____інформаційних технологій та програмної інженерії_____

Дисципліна ___Об’єктно-орієнтоване програмування_____

Спеціальність ___**121 «Інженерія програмного забезпечення»**__ _____

Курс_____II_____ Група _____ІПЗс-2019_____ Семестр _____4_____

ЗАВДАННЯ

на курсовий проект (роботу) студента

_____Ісарук Віталій Володимирович_____

Прізвище, ім’я, по-батькові

1. Тема проекту (роботи) Класовий тип для роботи з структурами типу «Вектор»_____

_____

2. Строк здачі студентом закінченого проекту (роботи) _____

_____

_____

3. Вихідні дані до проекту (роботи) _____

_____

_____

4. Зміст роботи (перелік питань по розділах, які потрібно розробити) _____

_____

_____

5. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсового проекту (роботи)	Строк виконання	Примітки
1	Ознайомлення з завданням та пошук інформації	10.05 – 11.05	
2	Розробка плану написання	13.05 – 14.05	
3	Розробка архітектури курсового проекту	15.05 – 16.05	
4	Написання коду	17.05 – 19.05	
5	Оформлення першої частини курсової	20.05 – 21.05	
6	Оформлення другої частини курсової	22.05 – 23.05	

Студент _____ _____

Підпис Прізвище, ініціали

Керівник _____ _____

Підпис Прізвище, ініціали

«____» _____ 202___ р.

ЗМІСТ

ВСТУП

Програмування — процес проектування, написання, тестування, зневадження і підтримки комп'ютерних програм. Програмування поєднує в собі елементи інженерії (існує навіть відповідна спеціальна галузь інженерії — програмна інженерія, англ. *software engineering*), фундаментальних наук (перш за все математики) і мистецтва. У вужчому значенні програмування розглядається як кодування — реалізація у вигляді програми одного чи кількох взаємопов'язаних алгоритмів (у сучасних умовах це здійснюється з застосуванням мов програмування). У широкому значенні програмування використовується у значенні створення програми дій або алгоритмів та навчання людей або пристроїв діяти за алгоритмами. Яким би не був підхід до створення програмного забезпечення, кінцева програма має задовольняти деяким вимогам.

Найчастіше зустрічаються:

- **Ефективність/Продуктивність**: кількість ресурсів системи, що споживає програма (час процесора, розмір пам'яті, зовнішня пам'ять, ширина каналу мережі, і навіть взаємодії з користувачем). Чим менше ресурсів споживається, тим краще;
- **Надійність**: ймовірність того, що результат роботи програми правильний. Це залежить від коректності алгоритмів та правильності кодування;
- **Стійкість**: як програма розв'язує проблеми в нестандартних ситуаціях, як наприклад неправильні дані, недоступність необхідних ресурсів як наприклад пам'ять, локальна мережа, та неправильні дії користувача;
- **Зручність**: ергономічність програми. Легкість, з якою особа може використовувати програму для своїх цілей;

- **Переносимість**: діапазон апаратного забезпечення та операційних систем на яких можна компілювати чи інтерпретувати код програми, виконуючи її. Це залежить від відмінностей в програмних ресурсах наданих різними платформами, включаючи ресурси, наявність компіляторів та бібліотек для мови програмування;
- **Масштабованість**: Простота подальшого супроводження програми, тобто внесення в неї додаткових вдосконалень, що збільшують функціональність чи виправляють помилки.
- **Естетичність**: Вигляд програми на екрані з точки зору підбору кольорів, форм, розмірів графічних елементів і контролів, гармонійності їх взаємного розташування, якість малюнків, вибір шрифтів тексту
- **Етичність**: Ступінь спрямованості на задоволення справжніх потреб людей, реалізацію кращих і законних бажань користувачів і розробників.

Про ООП.

Об'єктно-орієнтоване програмування, (ООП) — це метод програмування, заснований на поданні програми як сукупності взаємодіючих об'єктів, кожен з яких є примірником певного класу, а класи є членами певної ієрархії наслідуванн. Програмісти спочатку пишуть клас, а на його основі під час виконання програми створюються конкретні об'єкти (екземпляри класів). На основі класів можна створювати нові, які розширюють базовий клас і таким чином створюється ієрархія класів.

ООП, також являє собою, однією з парадигм програмування, що розглядає програму, як множину об’єктів, що взаємодіють між собою. А основою ООП,

складають чотири основні концепції: інкапсуляція, наслідування, поліморфізм та наслідування.

Однією з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-х роках, вона не мала широкого застосування до 1990-х, коли розвиток комп'ютерів та комп'ютерних мереж дав змогу писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Сьогодні багато мов програмування або підтримують ООП - (PHP, Lua) або ж є
цілком об'єктно-орієнтованими,
зокрема: Java, C#, C++, Python, Ruby і Objective-C, ActionScript 3, Swift, Vala.

Об'єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування Симула в 1960-х роках, одночасно з посиленням дискусій про кризу програмного забезпечення. Через ускладнення апаратного та програмного забезпечення було дуже важко зберегти якість програм. Об'єктно-орієнтоване програмування частково розв'язує цю проблему шляхом наголошення на модульності програми.
На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП-програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

Визначення про ООП.

На думку Алана Кея, розробника мови Smalltalk, якого вважають одним з «батьків-засновників» ООП, об'єктно-орієнтований підхід полягає в наступному наборі основних принципів:

- Все є об'єктами.
- Всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об'єктами, під час якої один об'єкт потребує, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, надсилаючи і отримуючи повідомлення. Повідомлення — це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися під час виконання дії.
- Кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів.
- Кожен об'єкт є представником (екземпляром, примірником) класу, який виражає загальні властивості об'єктів.
- У класі задається поведінка (функціональність) об'єкта. Таким чином усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії.
- Класи організовані у єдину деревоподібну структуру з загальним корінням, яка називається ієрархією успадкування. Пам'ять та поведінка, зв'язані з екземплярами деякого класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

Таким чином, програма є набором об'єктів, що мають стан та поведінку. Об'єкти взаємодіють використовуючи повідомлення. Будується ієрархія об'єктів: програма в цілому — це об'єкт, для виконання своїх функцій вона звертається до об'єктів що містяться у ньому, які у свою чергу виконують запит шляхом звернення до інших об'єктів програми. Звісно, щоб уникнути нескінченної рекурсії у зверненнях, на якомусь етапі об'єкт трансформує запит у повідомлення до стандартних системних об'єктів, що даються мовою та середовищем програмування. Стійкість та керованість системи забезпечуються за рахунок чіткого розподілу відповідальності об'єктів (за кожну дію відповідає певний об'єкт), однозначного означення інтерфейсів міжоб'єктної взаємодії та повної ізольованості внутрішньої структури об'єкта від зовнішнього середовища (інкапсуляції).

Історія ООП.

ООП - виникло в результаті розвитку ідеології процедурного програмування, де дані і підпрограми (процедури, функції) їх обробки формально не пов'язані. Для подальшого розвитку об'єктно-орієнтованого програмування велике значення мають поняття події (так зване подієво-орієнтоване програмування) і компоненти (компонентне програмування, КОП).

Формування КОП від ООП відбулося, так само як формування модульного від процедурного програмування: процедури сформувалися в модулі - незалежні частини коду до рівня збірки програми, так об'єкти сформувалися в компоненти - незалежні частини коду до рівня виконання програми. Взаємодія об'єктів відбувається за допомогою повідомлень. Результатом подальшого розвитку ООП, мабуть, буде агентно-орієнтоване програмування, де агенти - незалежні частини коду

на рівні виконання. Взаємодія агентів відбувається за допомогою зміни середовища, в якій вони знаходяться.

Мовні конструкції, конструктивно не пов'язані безпосередньо з об'єктами, але необхідні їм для їх безпечної (виняткові ситуації, перевірки) та ефективної роботи, інкапсулюються від них в аспекти (в аспектно - орієнтованому програмуванні). Суб'єктно-орієнтоване програмування розширює поняття об'єктів шляхом забезпечення більш уніфікованого і незалежної взаємодії об'єктів. Може бути перехідною стадією між ООП та агентним програмуванням в частині самостійної їх взаємодії.

Першою мовою програмування, в якій були запропоновані принципи об'єктної орієнтованості, була Симула. На момент своєї появи (в 1967 році), ця мова програмування запропонувала революційні ідеї: об'єкти, класи, віртуальні методи тощо, однак це все не було сприйнято сучасниками як щось грандіозне. Тим не менше, більшість концепцій були розвинені Аланом Кеєм та Деном Інгаллсом у мові Smalltalk. Саме вона стала першою широко поширеною об'єктно - орієнтованою мовою програмування.

Наразі кількість прикладних мов програмування (список мов), що реалізують об'єктно-орієнтовану парадигму, є найбільшим по відношенню до інших парадигм. В області системного програмування досі застосовується парадигма процедурного програмування, і загальноприйнятою мовою програмування є мова С. Хоча при взаємодії системного і прикладного рівнів операційних систем стали помітно впливати мови об'єктно-орієнтованого програмування. Наприклад, однією з найпоширеніших бібліотек мультиплатформового програмування є об'єктно-орієнтована бібліотека Qt, написана мовою С++.

Фундаментальні поняття.

В результаті дослідження Дебори Дж. Армстронг (англ. *Deborah J. Armstrong*), комп'ютерної літератури, що була видана протягом останніх 40 років, вдалось відокремити фундаментальні поняття (принципи), використані у переважній більшості визначень об'єктно-орієнтованого програмування.

До них належить:

Клас.

Клас визначає абстрактні характеристики деякої сутності, включно з характеристиками самої сутності (її **атрибутами** або **властивостями**) та діями, які вона здатна виконувати (її **поведінкою**, **методами** або **можливостями**). Наприклад, клас Собака може характеризуватись рисами, притаманними всім собакам, зокрема: порода, колір хутра, здатність гавкати. Класи вносять модульність та структурованість в об'єктно-орієнтовану програму. Зазвичай клас має бути зрозумілим для не-програмістів, що знаються на предметній області, що, у свою чергу, значить, що клас повинен мати значення в контексті. Також, код реалізації класу має бути досить самодостатнім. Властивості та методи класу, разом називаються його **членами**.

Об'єкт.

Окремий екземпляр класу (створюється після запуску програми і ініціалізації полів класу). Клас Собака відповідає всім собакам шляхом опису їхніх спільних рис; об'єкт Сірко є одним окремим собакою, окремим варіантом

значень характеристик. Собака має хутро; Сірко має коричнево-біле хутро. Об'єкт Сірко є **екземпляром** (**примірником**) класу Собака. Сукупність значень атрибутів окремого об'єкта називається **станом**. На основі класу Собака можна, також, створити інший об'єкт Дружок, який відрізнятиметься від об'єкта Сірко своїм станом (наприклад кольором хутра). Обидва об'єкта (Сірко і Дружок) є екземплярами класу Собака.

Метод.

Можливості об'єкта. Оскільки Сірко — Собака, він може гавкати.
Тому гавкати() є одним із методів об'єкта Сірко. Він може мати й інші методи, зокрема: місце(), або їсти(). В межах програми, використання методу має впливати лише на один об'єкт; всі Собаки можуть гавкати, але треба щоб гавкав лише один окремий собака.

Обмін повідомленнями.

«Передача даних від одного процесу іншому, або надсилання викликів методів.».

Успадкування (наслідування).

Клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть навіть утворюватись цілі дерева успадкування. Наприклад, клас Собака може мати підкласи Коллі, Пекінес, Вівчарка тощо. Так, Сірко може бути екземпляром класу Вівчарка. Підкласи *успадковують* атрибути та поведінку своїх батьківських класів, і можуть вводити свої власні. Успадкування може бути одиничне (один безпосередній батьківський клас) та множинне (кілька батьківських класів). Це залежить від вибору програміста, який реалізовує клас та мови програмування. Так, наприклад, в Java дозволене лише одинарне успадкування, а в C++ і те і інше.

Абстрагування.

Спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми. Наприклад Собака Сірко більшу частину часу може розглядатись як Собака, а коли потрібно отримати доступ до інформації специфічної для собак породи коллі — як Коллі і як Тварина (можливо, батьківський клас Собака) під час підрахунку тварин Петра.

Приховування інформації (інкапсуляція).

Приховування деталей про роботу класів від об'єктів, що їх використовують або надсилають їм повідомлення.

Так, наприклад, клас Собака має метод гавкати(). Реалізація цього методу описує як саме повинно відбуватись гавкання (приміром, спочатку вдихнути() а потім видихнути() на обраній частоті та гучності). Петро, хазяїн пса Сірка, не повинен знати як він гавкає. Інкапсуляція досягається шляхом вказування, які

класи можуть звертатися до членів об'єкта. Як наслідок, кожен об'єкт надає кожному іншому класу певний інтерфейс — члени, доступні іншим класам. Інкапсуляція потрібна для того, аби запобігти використанню користувачами інтерфейсу тих частин реалізації, які, швидше за все, будуть змінюватись. Це дасть змогу полегшити внесення змін без потреби змінювати і користувачів інтерфейсу. Наприклад, інтерфейс може гарантувати, що щенята можуть додаватись лише до об'єктів класу `Собака` кодом самого класу. Часто, члени класу позначаються як **публічні** (англ. public), **захищені** (англ. protected) та **приватні** (англ. private), визначаючи, чи доступні вони всім класам, підкласам, або лише до класу в якому їх визначено.

Деякі мови програмування йдуть ще далі: Java використовує ключове слово **private** для обмеження доступу, що буде дозволений лише з методів того самого класу, **protected** — лише з методів того самого класу і його нащадків та з класів із того ж самого пакету, C# та VB.NET відкривають деякі члени лише для класів із тієї ж збірки шляхом використання ключового слова **internal** (C#) або **Friend** (VB.NET), а Eiffel дозволяє вказувати які класи мають доступ до будь-яких членів.

Поліморфізм.

Поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по-різному на однакові повідомлення. Наприклад, якщо `Собака` отримує команду `голос()`, то у відповідь можна отримати `Гав`;

якщо Свиня отримує команду голос (), то у відповідь можна отримати Рох-рох. На практиці - це реалізовується шляхом реалізації ряду підпрограм (функцій, процедур, методи тощо) з однаковими іменами, але з різними параметрами. Залежно від того, що передається, і вибирається відповідна підпрограма.

Поняття про Java.

Java — об’єктно-орієнтова мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

«Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині.

Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

Java вплинула на розвиток J++, що розроблялась компанією «Microsoft». Роботу над J++ було зупинено через судовий позов «Sun Microsystems», оскільки ця

мова програмування була модифікацією Java. Пізніше в новій платформі «Microsoft» .NET випустили J#, щоб полегшити міграцію програмістів J++ або Java на нову платформу. З часом нова мова програмування C# стала основною мовою платформи, перейнявши багато чого з Java. J# востаннє включався в версію Microsoft Visual Studio 2005. Мова сценаріїв JavaScript має схожу із Java назву і синтаксис, але не пов'язана із Java.

Спочатку мова називалася Oak («дуб») і розроблялася Джеймсом Гослінгом для програмування побутових електронних пристроїв. Згодом вона була перейменована в Java і стала використовуватися для написання клієнтських застосунків і серверного програмного забезпечення. Названа на честь марки кави Java, яка, в свою чергу, отримала найменування однойменного острова (Ява), тому на офіційній емблемі мови зображена чашка з паркою кавою. Існує й інша версія походження назви мови, пов'язана з алюзією на каво-машину як приклад побутового устаткування, для програмування якого спочатку мова створювалася.

Історія Java.

Мова програмування Java зародилася в 1991 р. в лабораторіях компанії Sun Microsystems. Розробку проєкту започаткував Джеймс Ґослінґ, сам проєкт мав назву «Green» (Зелений). Створення першої робочої версії, яка мала назву «Oak» (дуб), зайняло 18 місяців. Оскільки виявилось, що ім'я Oak уже використовувалось іншою фірмою, то в результаті тривалих суперечок навколо назви нової мови з-поміж ряду

запропонованих було вибрано назву Java, у 1995 р. мову було офіційно перейменовано.

Головним мотивом створення Java була потреба в мові програмування, яка б не залежала від платформи (тобто від архітектури) і яку можна було б використовувати для створення програмного забезпечення, що вбудовується в різноманітні побутові електронні прилади, такі як мобільні засоби зв'язку, пристрої дистанційного керування тощо.

Досить скоро майже всі найпопулярніші тогочасні веб-оглядачі отримали можливість запускати «безпечні» для системи Java-аплети всередині веб-сторінок. У грудні 1998 р. Sun Microsystems випустила Java 2 (спершу під назвою J2SE 1.2), де було реалізовано декілька конфігурацій для різних типів платформ. Наприклад, J2EE призначалася для створення корпоративних застосунків, а значно урізана J2ME для приладів з обмеженими ресурсами, таких як мобільні телефони. У 2006 році в маркетингових цілях версії J2 було перейменовано у Java EE, Java ME та Java SE відповідно.

13 листопада 2006 року Sun випустили більшу частину Java як вільне та відкрите програмне забезпечення згідно з умовами GNU General Public License (GPL). 8 травня 2007 корпорація закінчила процес, в результаті якого всі початкові коди Java були випущенні під GPL, за винятком невеликої частини коду, на який Sun не мала авторського права.

Період становлення Java збігся у часі з розквітом міжнародної інформаційної служби World Wide Web. Ця обставина відіграла вирішальну роль у майбутньому Java, оскільки Web теж вимагала платформо-незалежних програм. Як наслідок, були зміщені акценти в розробці Sun з побутової електроніки на програмування для Інтернет.

Віртуальна машина Java.

Віртуальна машина Java, (англ. Java Virtual Machine, JVM) — набір комп'ютерних програм та структур даних, що використовують модель віртуальної машини для виконання інших комп'ютерних програм чи скриптів.

JVM використовує байт-код Java, який як правило, але не завжди генерується з вихідних кодів мови програмування Java; віртуальну машину також застосовують для виконання коду, згенерованого з інших мов програмування. Наприклад, вихідний код Ada можна скомпілювати у Java байт-код.

Віртуальна машина Java — основний компонент Java платформи. JVM доступна для всіх основних сучасних платформ, тому про програми, що скомпільовані у Java байткод теоретично можна сказати «Написано один раз, працює скрізь» (англ. "Write once, run anywhere").

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та роботу з мережами. У деяких версіях задля збільшення продуктивності JVM байт-код можна компілювати у машинний код до або під час виконання програми.

Основна перевага використання байт-коду — це портативність. Тим не менш, додаткові витрати на інтерпретацію означають, що інтерпретовані програми будуть майже завжди працювати повільніше, ніж скомпільовані у машинний код, і саме тому Java одержала репутацію «повільної» мови. Проте, цей розрив суттєво скоротився після введення декількох методів оптимізації у сучасних реалізаціях JVM.

Основним завданням розробників Java було створення переносите додатків. JVM грає центральну роль в переносимості - вона забезпечує належний рівень

абстракції між компільованою програмою і базової апаратної платформою і операційною системою. Незважаючи на цей додатковий «шар», швидкість роботи додатків надзвичайно висока, тому що байт-код, який виконує JVM, і вона сама чудово оптимізовані.

Складніші віртуальні машини також використовують динамічну рекомпіляцію, яка полягає в тому, що віртуальна машина аналізує поведінку запущеної програми й вибірково рекомпілює та оптимізує певні її частини. З використанням динамічної рекомпіляції можна досягти більшого рівня оптимізації, ніж за статичної компіляції, оскільки динамічний компілятор може робити оптимізації на базі знань про довкілля періоду виконання та про завантажені класи.

Java платформа.

Програмна платформа Java - ряд програмних продуктів і специфікацій компанії Sun Microsystems, раніше незалежної компанії, а нині дочірньої компанії корпорації Oracle, які спільно надають систему для розробки прикладного програмного забезпечення та вбудовування її в будь-який крос-платформенне програмне забезпечення.
Java використовується в самих різних комп'ютерних платформах від вбудованих пристроїв і мобільних телефонів в нижньому ціновому сегменті, до корпоративних серверів і суперкомп'ютерів у вищому ціновому сегменті**.**

Компілюючи початковий Java код у байт-код, який є спрощеними машинними командами. Потім програму можна виконати на будь-якій платформі, що має встановлену віртуальну машину Java, яка інтерпретує байткод у код, пристосований до специфіки конкретної операційної системи і процесора. Зараз віртуальні машини Java існують для більшості процесорів і операційних систем.

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та роботу з мережами. У деяких версіях задля збільшення продуктивності JVM байт-код можна компілювати у машинний код до або під час виконання програми.

Існує одна технологія оптимізації байткоду, широко відома як статична компіляція, або компіляція ahead-of-time (AOT). Цей метод передбачає, як і традиційні компілятори, безпосередню компіляцію у машинний код. Це забезпечує хороші показники в порівнянні з інтерпретацією, але за рахунок втрати переносності: скомпільовану таким способом програму можна запустити тільки на одній, цільовій платформі.

Швидкість офіційної віртуальної машини Java значно покращилася з моменту випуску ранніх версій, до того ж, деякі випробування показали, що продуктивність JIT-компіляторів у порівнянні зі звичайними компіляторами у машинний код майже однакова. Проте ефективність компіляторів не завжди свідчить про швидкість виконання скомпільованого коду, тільки ретельне тестування може виявити справжню ефективність у даній системі.

Технологія Java-аплетів стала рідко використовуваної в настільних комп'ютерах, проте вона іноді використовується для поліпшення функціональності і підвищення безпеки при перегляді всесвітньої павутини. Програмний код, написаний на Java, віртуальна машина Java виконує байт-код Java. Однак є компілятори байт-коду для інших мов програмування, таких як: Ada, JavaScript, Python, і Ruby. Також є кілька нових мов програмування, розроблених для роботи з віртуальною машиною Java.

Java «фреймворки».

Фреймворк — інфраструктура програмних рішень, що полегшує розробку складних систем. Спрощено дану інфраструктуру можна вважати своєрідною комплексною бібліотекою, але при цьому вона має ряд обмежень, що задають правила створення структури проєкту та написання коду.

Програмний фреймворк (— це готовий до використання комплекс програмних рішень, включаючи дизайн, логіку та базову функціональність системи або підсистеми. Відповідно — програмний фреймворк може містити в собі також допоміжні програми, деякі бібліотеки коду, скрипти та загалом все, що полегшує створення та поєднання різних компонентів великого програмного забезпечення чи швидке створення готового і не обов'язково об'ємного програмного продукту. Побудова кінцевого продукту відбувається, зазвичай, на базі єдиного API.

9 найпопулярніших Java «фреймворків»:

1) Spring.

Посідає перше місце з огляду на його підвищеної зручності в розробці комплексних web-додатків, що вимагають високої якості. З його допомогою розробники з легкістю створюють програмні рішення для великих підприємств. Фактично саме це зручність і функціональність роблять його найпопулярнішим в середовищі Java-розробки. В якості підтвердження нижче приведена наочна статистика використання різних фреймворків. Розробники істотно частіше

вибирають Spring MVC і Spring Boot. Найбільшою перевагою цих фреймворків є можливість від'єднання інших модулів і зосередження на одному завдяки інверсії управління (IoC).

До інших переваг цього фреймворка можна віднести повноцінну модель конфігурації, підтримку усталених і новітніх баз даних, таких як NoSQL, а також цілісність процесу розробки, підтримуючи аспектно-орієнтоване програмування. Він включає в себе такі модулі, як Spring MVC, Spring Core, Spring Boost, SpringTransaction тощо.

2) Hibernate.

Цей фреймворк - ORM, об'єктно-реляційного відображення, тобто він дозволяє писати запити до сервера бази даних не на SQL, а на Java, що змінює звичний погляд на базу даних як таку. Незважаючи на те, що Hibernate не є повноцінним фреймворком, він дозволяє з легкістю конвертувати інформацію для різних баз даних.

Ця особливість також спрощує масштабування, незалежно від розміру програми і кількості його користувачів. В цілому, цей фреймворк можна охарактеризувати як швидкий, потужний, легко масштабується і настроюється.

3) Struts.

Struts дозволяє розробляти для підприємств простий і зручний у використанні софт, де основною перевагою цього фреймворку виступають його портативні плагіни, які є пакетами JAR. Модулі Hibernate і Spring в цьому випадку можуть використовуватися для об'єктно-реляційного відображення і впровадження залежностей, відповідно. Цей фреймворк також дозволяє скоротити загальний час розробки за рахунок вдалої організації Java, JSP і Action класів.

4) Play.

Його застосовують такі найбільші компанії як LinkedIn, Samsung, The Guardian, Verizon і інші, що підтверджує його безумовну надійність. До основних відмінних характеристик можна віднести високу швидкість, якість і гарну масштабованість.

Інтерфейс Play є дуже простим і освоюється розробниками мобільних додатків досить швидко. Застосування ж він найчастіше знаходить в тих додатках, які вимагають регулярного створення контенту.

5) Google Web Toolkit.

Цей безкоштовний фреймворк застосовується для розробки клієнтської частини додатків, наприклад в Javascript. Корпорація Google широко застосовувала його в створенні великої кількості своїх сервісів, включаючи AdSense, Google Wallet, AdWords тощо.

За допомогою його кодів можуть бути також легко розроблені і налагоджені додатки Ajax. Розробники вибирають цей фреймворк при написанні комплексних програм, а його основні фішки - це крос-браузерна сумісність, зберігання історії, можливість ставити мітки та ін.

6) Grails.

Цей фреймворк також є безкоштовним і досить популярний в роботі з Enterprise Java Beans. Він може застосовуватися для створення надійних, масштабованих додатків, систем управління контентом, RESTful сервісів і комерційних сайтів.

Grails можна застосовувати разом з іншими технологіями Java - Spring, Hibernate, quartz, SiteMesh і контейнерами EE. Його переваги проявляються у вигляді наявності GORM, гнучких профілів, просунутої системи численних плагінів і бібліотеки відображення об'єктів.

7) Blade.

Практично будь-який розробник додатків зможе розібратися в цьому фреймворку протягом дня. Java Blade був випущений в 2015 і відразу став відомий як простий і легкий інструмент. Найяскравішим його перевагою є можливість дуже швидко створювати web-додатки.

Blade відноситься до повноцінних фреймворками, що пропонують просту і ясну структуру написання коду, підтримку web jar ресурсів і плагінів. Заснований він на Java 8, завдяки чому має інтерфейс маршрутизації в стилі RESTful.

8) Java Server Faces.

Цей фреймворк був впроваджений в індустрію компанією Oracle. JSF можна використовувати для створення корпоративних додатків, нативних програм, а також в web-розробці. Головна особливість полягає в можливості легко пов'язувати рівень уявлення з кодом програми.

Його API набір застосовується для подання і управління компонентами UI. JSF також має ясну архітектуру, яка різниться між логікою програми та його поданням. Ще однією відмінністю є уявлення за допомогою XML, а не Java.

9) Vaadin.

Відмінна платформа для впорядкованої розробки. Великою перевагою цього фреймворка є плавне взаємодія між сервером і браузером.Vaadin надає доступ до DOM безпосередньо з віртуальної машини Java. В останньому релізі його розділили на дві частини і перейменували в Vaadin Flow. Однак він як і раніше є легковажним фреймворком, що здійснює комунікацію і маршрутизацію на стороні сервера.

Класові типи для роботи з «векторами».

В ООП, клас — це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій. При цьому, згідно з термінологією ООП, глобальні змінні класу називаються полями даних, властивостями або атрибутами. Створений та ініціалізований екземпляр класу називають об'єктом класу. На основі одного класу можна створити безліч об'єктів, що відрізнятимуться один від одного своїм станом (значеннями полів).

Типи даних.

Java — строго типізована мова. Тобто, на відміну від, наприклад, PHP в Java потрібно вказувати тип змінної при її оголошенні. В Java є вісім основних (примітивних) типів даних.

П'ять із них — цілочисельні (включаючи і символьний тип char), два — дійсні (float, double), та один логічний (булевий) тип даних.

Числові типи складаються із цілих типів: byte, short, int, long.

Поняття про класи.

Класи – це конструкції спеціального виду, які дозволяють об’єднати ряд змінних різних типів в одне ціле. Крім власне даних, класи зазвичай включають підпрограми (в термінології java - методи) і можуть включати блоки (сукупність інструкцій між фігурними дужками {}) та інші класи (внутрішні класи). Таким чином утворюються нові типи даних.

Роз’яснення принципів побудови і роботи з класами доволі громіздка тема, тому тут розглянемо лише спрощені основи їх роботи і створення. Детальніше дивіться розділ присвячений класам. Розглянути зараз необхідно через те, що Java повністю об’єктно-орієнтована мова. З класами та пов’язаними з ними методами ви зіштовхуватиметися на кожному кроці. Зокрема, навіть виведення даних System.out.println(); - здійснюється за допомогою класу System та методу println().

Поля класу дозволяють вмістити дані про певний реальний об'єкт, а методи здійснювати обробку цих даних. Наприклад, можна створити загальний клас Людина з полями Ім'я та Прізвище, рік народження, професія, зарплата. При створенні ж на основі класу конкретного екземпляру, дані поля заповнюються конкретними даними про певну людину. Обробкою цих даних можуть займатися відповідні методи. Наприклад, можна створити метод для обчислення віку людини, тощо.

На основі класів можна створювати підкласи, які успадковують властивості та поведінку батьківських класів. Таким чином можна створити цілу ієрархію класів. Різні мови дещо по різному реалізовують механізм успадкування. Існує множинне та одинарне успадкування. Множинне — це, коли підклас створюється на основі кількох безпосередніх батьків (як то в мові програмування C++). Одинарне успадкування — це коли клас може мати одного безпосереднього батька (мова програмування Java). Надкласи можуть мати свої надкласи, підкласи можуть також бути надкласами для певних класів.

Поняття про «вектори».

Вектор — у найпростішому випадку математичний об'єкт, який характеризується величиною і напрямком. Наприклад, у геометрії і в природничих науках вектор є спрямований відрізок прямої в евклідовому просторі (або на площині).

Приклади: радіус-вектор, швидкість, момент сили. Якщо в просторі задана система координат, то вектор однозначно задається набором своїх координат. Тому в математиці, інформатиці та інших науках упорядкований набір чисел часто теж називають вектором. У більш загальному сенсі вектор у математиці розглядається як елемент деякого векторного (лінійного) простору.

Є одним з основоположних понять лінійної алгебри. При використанні найбільш загального означення векторами виявляються практично всі досліджувані в лінійній алгебрі об'єкти, зокрема матриці, тензори, однак, за наявності в навколишньому контексті цих об'єктів, під вектором мають на увазі відповідно вектор-рядок або вектор-стовпець, тензор першого рангу. Властивості операцій над векторами вивчаються у векторному численні.

Приклад 1:

$\langle a_1, a_2, \ldots, a_n \rangle,\ (a_1, a_2, \ldots, a_n), \{a_1, a_2, \ldots, a_n\}$.

Приклад 2:

$\bar{a},\ \vec{a}, \mathbf{a}, \mathfrak{A},\ \mathfrak{a}.$

Приклад 3:

$\vec{a} + \vec{b}$.

Робота з структурою типу вектор.

Параметризовані класи.

Узагальнення – це механізм побудови програмного коду для деякого типу з довільним іменем з метою його подальшого конвертування (перетворення) у будь-який конкретний посилальний тип. Реалізацію конвертування з узагальненого типу в деякий конкретний здійснює компілятор. Узагальнення можуть бути застосовані до класів, інтерфейсів чи методів. Якщо клас, інтерфейс чи метод оперує деяким узагальненим типом T, то цей клас (інтерфейс, метод) називається узагальненим. Тип, що отримує узагальнений клас в якості параметру називається параметризованим типом. Ім’я параметризованого типу можна задавати будь-яким (T, Type, TT і т.д.).

Динамічний масив.

Динамічний масив - це масив, який вміє змінювати свій розмір під час виконання програми. В Java цю роль грають в основному класи ArrayList і LinkedList. На відміну від масивів, ArrayList і LinkedList містять тільки посилальні типи даних, тобто зберегти в них можна тільки об'єкти. На щастя, в Java існують механізми автоупаковкі і автораспаковкі, які дозволяють зберігати в динамічних масивах примітивні типи. Подібно до статичної масиву, динамічний однорідний, тобто має здатність зберігати один-єдиний тип даних.

Однак, завдяки механізму наслідування і грамотного використання інтерфейсів, можна зберігати в одному динамічному масиві цілий спектр різноманітних класів, які успадковані від одного загального, але про це нижче.

Завдання до курсової роботи.

1. Створити параметризований клас для роботи з структурою типу Vector - одновимірний динамічний масив із змінною кількістю елементів. Тип елементу масиву надається параметром шаблону.

2. Передбачити конструктор для створення вектору та ініціалізації його звичайним одновимірним масивом відповідного типу.

3. Реалізувати компонентні методи:

- get - доступу до елемента за індексом;
- concat - конкатенації двох векторів;
- equals - порівняння двох векторів.
- size - кількість елементів вектора;
- front - посилання на перший елемент;
- back - посилання на останній елемент;
- swap - обмін значеннями з іншим вектором;
- insert - уставити елемент у надану позицію;
- push_back - додати новий елемент у кінець вектора;
- pop_back - вилучити останній елемент;
- erase - вилучити елемент у наданій позиції;
- find - знайти елемент у векторі і повернути його позицію;
- accumulate - накопичення суми або добутку;
- for_each - обробка елементів по наданій процедурі;
- max, min - пошук максимального та мінімального елементів;
- sort - упорядкування елементів у порядку зростання або зменшення.

Методи та функції в даній програмі програмі.

1) public class Vector<T extends Number> – функція основного інтерфейсу програми. Створення параметризованого класу Vector з абстрактним типом даних T і наслідуванням інтерфейсу Number.
2) public T get(int id) – функція , приймаюча id-індекс елемента динамічного масиву «Vector», з типом даних «T».
3) public boolean equals(Vector<T> vector2) – логічний, (булевий) метод, що порівнює між собою значення двох векторів, та повертає логіну відповідь.
4) public T front() – функція переходу, посилання на перший елемент в масиві.
5) public T beck() – функція переходу, посилання на останній елемент в масиві.
6) public void insert(int id, T element) – функція підстаноки елементу у надану позицію, що приймає id-індекс, а також елемент масива «Vector» з типом даних «T».
7) public void swap(Vector<T> vector2, int id) – метод, обміну значеннями з іншим вектором, приймає елемент масиву «Vector» з типом даних «T» та обмінується значенням id-індексу з «vector2».
8) public void push_back(T element) – метод додавання нового елемента в кінець вектора з типом даних «T».
9) public void pop_back(T element) – метод вилучення останнього елемента.
10) public void erase(int id) – метод вилучення елемента з обраної позиції.
11) public int find(T element) – метод пошуку елемента у векторі.
12) public int accumulate() – метод накопичення суми.
13) public void forEach() – функція обробки елементів.
14) public void concat() – функція конкатенації двох векторів.

Написання програми.

Отож, створивши новий клас, з назвою «Vector», та підключивши списки «ArrayList», «List» та клас «Arrays» для роботи з масивами, приступаю до написання самого коду, функцій та методів.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
```

Створення векторів: «vector1» та «vector2».

```
public class Vector<T extends Number> {
    private List<T> vector1 = new ArrayList<>(5);
    private List<T> vector2 = new ArrayList<>(3);
    private List<T> concatenating = new ArrayList<>();
    public Vector(T param) {
        this.vector1.addAll(Arrays.asList(param));
    }
```

// У рядку public Vector(T param) – задаються параметри саме для «vector1», як динамічний масив.

Написання методів та функцій програми.

// №1. Метод "get()".

```
public T get(int id) {
    return vector1.get(id);
}
```

Даний метод приймає id-індекс елемента динамічного масиву «Vector», з типом даних «T».

// ----------

// №2. Метод “size()”.

```
public int size() {
    return vector1.size();
}
```

Даний метод, повертатиме елементи динамічного масиву

// ----------

// №3. Метод “equals()”.

```
public boolean equals(Vector<T> vector2) {
    if(vector2 == null) {return false;}
    if(vector2.size() != vector1.size()) {return false;}
    for(int i = 0; i < vector1.size(); i++) {
        if(!vector1.get(i).equals(vector2.get(i))) {return
false;}
    } return true;
}
```

Даний метод порівнює вектори, та повертає логічну відповідь при заданих умовах.

// ----------

// №4. Метод “front()”.

```
public T front() {
    return vector1.isEmpty() ?
            null: vector1.get(0);
}
```

Даний метод посилається на перший елемент, якщо він не є пустим.

// -

// №5. Метод “back()”.

```
public T back() {
    return vector1.isEmpty() ?
            null: vector1.get(vector1.size() -1);
}
```

Даний метод переходить на останній елемент, а «-1» потрібно для того, щоб в майбутньому цикли не додавали і ще один елемент в кінець масива.

// -

// №6. Метод “insert()”.

```
public void insert(int id, T element) {
    if(id > 0 && id < vector1.size()) {
        vector1.set(id, element);
    }
}
```

Даний метод підставляє елементу у надану позицію, що приймає id-індекс та «T» елемент.

// -

// №7. Метод “swap()”.

```
public void swap(Vector<T> vector2, int id) {
    if(vector2.size() > id && vector1.size() > id) {
        T swapper = vector2.get(id);
        vector2.insert(id, vector1.get(id));
        vector1.set(id, swapper);
    }
}
```

Даний метод обмінюється значеннями з іншим вектором, приймає елемент масиву «Vector» з типом даних «T» та обмінується значенням id-індексу з «vector2».

// -

// №8. Метод “push_back()”.

```
public void push_back(T element) {
    vector1.add(element);
}
```

Даний метод додає новий елемент в кінець вектора з типом даних «T».

// -

// №9. Метод “pop_back()”.

```
public void pop_back(T element) {
    vector1.remove(element);
}
```

Даний метод видаляє останній елемент.

// -

//№10. Метод “erase()”.

```
public void erase(int id) {
    if(id < vector1.size()) {
        vector1.remove(vector1.get(id));
    }
}
```

Даний метод вилучає елемент з обраної позиції.

// -

//№11. Метод “find()”.

```
public int find(T element) {
    for(int i = 0; i < vector1.size(); i++) {
        if(vector1.get(i).equals(element)) {return i;}
    } return -1;
}
```

Даний метод знаходить елемент у векторі.

//----------

//№12. Метод “accumulate()”.

```
public int accumulate() {
    int sum = 0;
    for(int i = 0; i < vector1.size(); i++) {
        sum += vector1.get(i).intValue();
    } return sum;
}
```

Даний метод накопичує суму.

//----------

//№13. Метод “forEach()”.

```
public void forEach() {
    vector1.forEach(element -> System.out.println(element));
}
```

Даний метод оброблятиме елементи по вказаній процедурі.

//----------

//№14. Метод “concat()”.

```
public void concat() {
    concatenating.addAll(vector1);
    concatenating.addAll(vector2);
    System.out.println("Конкатенація векторів: " +
concatenating);
}
```

Даний метод конкатенує вектора: «vector1» та «vector2».

//----------

ВИСНОВОК

За час виконання даної курсової роботи, були закріплення знання про Об’єктно-орєнтоване програмування, його парадигми та фуедаментальні поняття. Закріплено знання про мову програмування Java, а саме: що таке Java, віртуальна машина Java, та її платформа, а також, популярні Java «фреймворки».

Також, було написано програму параметризованого класу для роботи з структурою типу «Вектор», де були освоєні та закріплені НОВІ знання роботи з масивами, а в особливості, з динамічним масивом, він же - «Вектор».

Повний код даної програми, а також декілька знімків екрану з результатами виводу методів в терміналі закріплені нижче, у розділі: «**Додаток**», та список використаних джерел під час виконання курсової роботи, закріплені нижче, у розділі: «**Список використаних джерел**».

ДОДАТОК

Знімки екрану про вивід методів в терміналі, та повний код програми

Вивід методів.

Для прикладу, щоб показати що методи працюють, нижче прикріплю декілька результатів виводу в терміналі.

// -

// №1. Метод "get()".

```
Vector<Integer> Vector1 = new Vector<>(5);
Vector<Integer> Vector2 = new Vector<>(3);
System.out.println("Getting vector: " + Vector1.get(0));
System.out.println("- - - - - - - - - - - -");
```

Результат виводу:

```
/Users/isarukvitalii/Library/Java/JavaVirtualMachines/
Getting vector: 5
- - - - - - - - - - - -


Process finished with exit code 0
```

«Vector1», (як і «Vector2») – є динамічними, а значить, які б параметри при сворені не було задані, він прийме їх.

// -

// №2. Метод “size()”.

```
Vector<Integer> Vector1 = new Vector<>(5);
   Vector<Integer> Vector2 = new Vector<>(3);
System.out.println("Vector size: " + Vector1.size());
System.out.println("- - - - - - - - - - - -");
```

Результат виводу:

```
/Users/isarukvitalii/Library/Java/JavaVirtualMachines/
Vector size: 1
- - - - - - - - - - -

Process finished with exit code 0
```

По стандарту, довжина динамічного масиву «Вектор» дорівнює 1.

// -

// №3. Метод “equals()”.

```
Vector<Integer> Vector1 = new Vector<>(5);
Vector<Integer> Vector2 = new Vector<>(3);
    System.out.println("Vectors equals: " +
Vector1.equals(Vector2));
    System.out.println("- - - - - - - - - - -");
```

Результат виводу:

```
/Users/isarukvitalii/Library/Java/JavaVirtualMachines/
Vectors equals: false
- - - - - - - - - - -

Process finished with exit code 0
```

Результат дорівнює «false», так як задан параметри двох векторів – різні.

// -

// №4. Метод “front()”.

```
Vector<Integer> Vector1 = new Vector<>(5);
Vector<Integer> Vector2 = new Vector<>(3);

System.out.println("Go to first element: " +
Vector1.front());
System.out.println("- - - - - - - - - - -");
```

Результат виводу:

```
/Users/isarukvitalii/Library/Java/JavaVirtualMachines/
Go to first element: 5
- - - - - - - - - - -

Process finished with exit code 0
```

Результат дорівнює 5 – так як кількість елементів в динамічному масив по стандарту дорівнює 1, а значення задається при створені і динамічний масив його приймає.

// -

// №5. Метод “back()”.

```java
Vector<Integer> Vector1 = new Vector<>(5);
Vector<Integer> Vector2 = new Vector<>(3);

System.out.println("Go to last element: " + Vector1.back());
System.out.println("- - - - - - - - - - -");
```

Результат виводу:

```
/Users/isarukvitalii/Library/Java/JavaVirtualMachines/
Go to last element: 5
- - - - - - - - - - -

Process finished with exit code 0
```

Результат знову ж таки дорівнює тому параметру, що задається на початку, тому що стандартна «довжина» динамічного масиву дорівнює 1.

//----------

//№6. Метод “concat()”.

```
Vector<Integer> Vector1 = new Vector<>(5);
Vector<Integer> Vector2 = new Vector<>(3);
Vector1.concat();
Vector2.concat();
```

Результат виводу:

```
/Users/isarukvitalii/Library/Java/JavaVirtualMachines/
Конкатенація векторів: [5]
Конкатенація векторів: [3]

Process finished with exit code 0
```

Конкатенація векторів: «vector1» та «vector2».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Посилання на використані джерела:

1) Vertex Academy: https://vertex-academy.com/tutorials/ru/arraylist-v-java/
2) JavaRush: https://javarush.ru/groups/posts/dinamicheskie-massivy-java
3) WikiBooks: https://uk.wikibooks.org/
4) Wikipedia: https://uk.wikipedia.org/wiki/
5) QA Group: https://qagroup.com.ua/
6) NOP: https://nuancesprog.ru/p/5857/
7) Metanit: https://metanit.com/java/tutorial/3.1.php
8) BestProg: https://www.bestprog.net/
9) tProger: https://tproger.ru/blogs/jvm-insides/
10) Java & PHP: http://javaphp.ptngu.com/phplessons/phplesson2-1

м. Івано-Франківськ

2021