

Методичні вказівки до виконання комп'ютерних
практикумів

Програмування, частина II

Об'єктно- орієнтоване програмування



Системна інженерія

Кафедра Технічної Кібернетики
НТУУ “КПІ”

Київ 2015

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ"
Кафедра технічної кібернетики

"ЗАТВЕРДЖУЮ"

Декан ФІОТ О.А.Павлов

« _____ » _____

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ СТУДЕНТАМ ЩОДО ЗАСВОЄННЯ
КРЕДИТНОГО МОДУЛЯ

"ПРОГРАМУВАННЯ, ч.ІІ
ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ "

для напрямку підготовки 6.050201 "Системна інженерія"

Рекомендовано кафедрою
технічної кібернетики
протокол № 7

від "28" січня 2015р.

Завідувач кафедри

М.М. Ткач

Програмування, ч.2. Об'єктно-орієнтоване програмування: Метод. вказівки до комп'ютерного практикуму для студентів 1-го курсу напряму «Системна інженерія» Уклад. Р.І., Дзінько, О.І., Лісовиченко. – К.: НТУУ-КПІ, 2015 – 32с.

*Рекомендовано кафедрою
технічної кібернетики ФІОТ НТУУ “КПІ”
(Протокол №7 від 28.018.2015 р.)*

Навчальне видання

**Програмування, ч.2
Об'єктно-орієнтоване програмування**

**МЕТОДИЧНІ ВКАЗІВКИ
ДО КОМП'ЮТЕРНОГО ПРАКТИКУМУ
для студентів 1-го курсу напряму підготовки 6.050201
«Системна інженерія»**

Укладачі	<i>Дзінько Ростислав Ігорович, асистент Лісовиченко Олег Іванович, к.т.н., доцент</i>
Відповідальний редактор	<i>Ткач Михайло Мартинович, к.т.н., доцент</i>
Рецензенти	<i>Павлов Олександр Анатолійович, д.т.н., професор</i>

Вступ

Комп'ютерні практикуми з модуля «Об'єктно-орієнтоване програмування» виконуються студентами I курсу напряму підготовки 6.0914 «Системна інженерія» спеціальність 7.091402 «Гнучкі комп'ютеризовані системи та робототехніка» денної та заочної форми навчання.

Основна мета комп'ютерного практикуму – закріпити знання, одержані на лекційних, практичних заняттях, та в результаті самостійної роботи. Студенти опановують засоби об'єктно-орієнтованої розробки на мові програмування C++, розвивають навички складання програм та об'єктно-орієнтованого проектування, освоюють прийоми організації типових компонентів додатків.

В якості інструментів для навчання C++ використовується набір інструментів Qt Framework 5, яке включає не лише відповідний компілятор, але і текстовий редактор, компонувальник, відлагоджувач, редактор інтерфейсу, та інші засоби, що прискорюють процес підготовки програми.

Фреймворк Qt 5, як один із засобів автоматизації програмування є крос-платформним програмним засобом, що виконується у всіх популярних операційних системах (Windows, Linux, Unix, Mac OS X, iOS, Android, і т.д.).

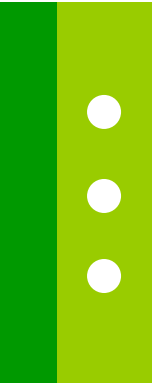
При роботі в лабораторії інформаційних технологій необхідно вживати заходів для боротьби з програмними вірусами та дотримуватись правил роботи та техніки безпеки.

Зміст

Вступ	4
Зміст	5
Комп'ютерний практикум №1	6
<i>Короткі теоретичні відомості</i>	7
Оцінка складності алгоритму.....	7
Динамічні масиви	9
Однозв'язні списки	10
<i>Завдання до виконання</i>	11
<i>Контрольні запитання</i>	12
Комп'ютерний практикум №2	13
<i>Короткі теоретичні відомості</i>	14
Архітектура графічних додатків	14
Мінімальна графічна програма на Qt.....	15
Шаблони проектування ПЗ	16
Singleton	17
<i>Завдання до виконання</i>	18
<i>Контрольні запитання</i>	19
Комп'ютерний практикум №3	21
<i>Короткі теоретичні відомості</i>	22
Шаблони проектування «Стан» та «Стратегія».....	22
Організація журналу додатку	24
<i>Завдання до виконання</i>	24
<i>Контрольні запитання</i>	25
Комп'ютерний практикум №4	26
<i>Короткі теоретичні відомості</i>	27
Шаблон проектування «Фабричний метод»	27
Шаблон проектування «Шаблонний метод»	27
Організація ігрового процесу.....	28
<i>Завдання до виконання</i>	28
<i>Контрольні запитання</i>	29
Комп'ютерний практикум №5	30
<i>Короткі теоретичні відомості</i>	31
Standard Templates Library (STL).....	31
<i>Завдання до виконання</i>	31
<i>Контрольні запитання</i>	32

Комп'ютерний практикум №1

Основи використання класів

- 
- *Базові структури даних та алгоритми*
 - *Конструктори, деструктори*
 - *Класи пам'яті, управління пам'яттю*

Короткі теоретичні відомості

Оцінка складності алгоритму

Алгоритм (algorithm)

Послідовність дій (кроків), що призводить до отримання певного результату на основі поданих вхідних даних.

Властивості алгоритмів:

Дискретність	<i>алгоритм складається з чітко розділених кроків</i>
Скінченність	<i>кількість кроків алгоритму є кінцевою і завчасно визначеною</i>
Зрозумілість	<i>алгоритм повинен складатись з кроків, зрозумілих його виконавцю</i>
Детермінованість	<i>кожна команда алгоритму є однозначною і не потребує додаткових джерел для свого тлумачення</i>
Формальність	<i>властивість алгоритму давати однаковий результат незалежно від виконавця</i>
Масовість	<i>здатність алгоритму бути застосованим не до однієї конкретної задачі, а до цілого класу задач, що визначаються областю визначення вхідних даних</i>

Обчислювальна складність алгоритму (Calculation complexity)

Це оцінка часу обчислень, що необхідний для виконання всіх кроків алгоритму.

Нехай A - деякий алгоритм, що вирішує певний клас задач, а N - розмірність окремої задачі (наприклад, для задачі сортування масиву, N буде рівним

кількості елементів масиву, для задач пошуку в графах, N буде рівним кількості вершин графу і т.д.). Введемо поняття деякої функції $f_A(N)$, що повертає верхню межу кількості основних операцій (наприклад, множення, додавання, і т.д.), що виконуються для вирішення основної задачі розмірності N .

В такому випадку, якщо зі зростанням N , значення функції $f_A(N)$ зростає не швидше, ніж значення N , кажуть, що алгоритм A – **поліноміальний**, якщо швидше – експоненціальний.

Розглянемо оцінювання часової складності алгоритму. Дана оцінка виконується шляхом обчислення величини O - «велике O » (eng. «Big- O »). Кажуть, що функція $f_A(N)$ зростає як $g(N)$ для великих значень N , що формально подається як $f_A(N) = O(g(N))$. У випадку, якщо існує певна невід’ємна константа $Const > 0$, така, що $\lim_{N \rightarrow \infty} \frac{f_A(N)}{g(N)} = 0$, то оцінка $O(g(N))$ називається **асимптотичною часовою складністю алгоритму**.

Така оцінка застосовується, коли точне значення часу $f_A(N)$ є невідомим, а можна визначити лише порядок зростання часу зі зростанням розмірності задачі.

Приклади асимптотичних складностей:

Складність	Опис	Приклад
$O(1)$	Сталий час роботи, незалежно від розмірності задачі	Очікуваний час пошуку значення в хеш-таблиці
$O(\log N)$	Дуже повільне зростання часу	Інтерполяційний пошук в масиві
$O(\log \log N)$	Логарифмічна складність – подвоєння розміру задачі збільшує час роботи на сталу величину	Двійковий пошук в масиві
$O(N)$	Лінійне зростання	Лінійний пошук в масиві
$O(N \log N)$	Лінеаритмічне зростання – подвоєння розміру задачі збільшує час трохи більше, ніж вдвічі	Сортування злиттям
$O(N^2)$	Квадратичне зростання	Сортування бульбашкою
$O(N^3)$	Кубічне зростання	Множення матриць

Динамічні масиви

Динамічний масив (dynamic array)

Масив, розмір якого може змінюватись в процесі роботи програми (*runtime*)

Навідміну від статичного масиву володіє такими властивостями як розмір (*size*) – кількість заповнених елементів масиву, та ємність (*capacity*) – кількість елементів масиву, під які фізично виділено пам'ять.

Динамічний масив в C++ реалізується за допомогою шаблону **std::vector**, проте в рамках даної лабораторної роботи, виконати реалізацію динамічного масиву цілих чисел доведеться власноруч, створенням відповідного класу.

Інтерфейс динамічного масиву повинен бути наступним:

- *void pushBack(int element)* – додавання цілого числа в кінець масиву
- *void insert(int pos, int element)* – вставка цілого числа *element* за індексом *pos*.
- *void pushFront(int element)* – додавання цілого числа на початок масиву
- *void deleteAt(int pos)* – видалити елемент з індексом *pos*.
- *void reverse()* – перестановка елементів масиву в оберненому порядку
- *std::string toString()* – створення рядкового представлення масиву
- *void display()* – вивід масиву в термінал
- *unsigned int getSize()* – повернення розміру масиву (логічного)
- *unsigned int getCapacity()* – повернення ємності масиву (фізичної)
- *void sort()* – сортування масиву бульбашкою
- *int elementAt(int pos)* – отримання значення елемента з індексом *pos*.

В C++ динамічне виділення пам'яті здійснюється за допомогою оператора **new**. Наприклад:

- **int *p_x = new int;** // виділяє пам'ять та викликає конструктор по замовчуванню
- **int *p_x = new int(5);** // виділяє пам'ять під новий об'єкт типу **int** та ініціалізує ціле число значенням **5**
- **int *p_ar = new int[5];** // виділяє пам'ять під масив з **5** цілих чисел

Звільнення виділеної пам'яті здійснюється за допомогою оператора **delete**. Наприклад:

- **delete p_x;** // звільняє пам'ять, виділену під об'єкт **p_x**, де **p_x** – вказівник

- `delete[] p_ar;` // звільняє пам'ять, виділену під масив

Однозв'язні списки

Однозв'язний список (Linked List)

Це структура даних, що зберігає колекцію об'єктів, проте, на відміну від масиву, елементи в списку не зберігаються в послідовних комірках пам'яті.

Елементами списку є об'єкти, що складаються з двох полів:

- Значення елемента списку
- Вказівника на наступний елемент списку

Сам об'єкт списку повинен зберігати показчик на перший елемент списку.

На відміну від динамічного масиву, пам'ять під елементи списку виділяється не завчасно, а в момент додавання нового елемента списку. Вказівник останнього елемента списку має значення **NULL**.

В С++ список реалізований стандартним типом **std::list**, проте в рамках даної лабораторної роботи, виконати реалізацію списку цілих чисел доведеться власноруч.

Інтерфейс списку повинен бути ідентичним інтерфейсу динамічного масиву.

Завдання до виконання

- Виконання завдання певного рівня також включає в себе виконання завдань всіх попередніх рівнів.
- Рівні йдуть по порядку по мірі зростання складності.
- Завдання включають матеріал 1-го та 2-го розділів конспекту лекцій.
- Рівень *GOD** можна реалізовувати окремо, без реалізації завдань попередніх рівнів.

Рівень E

1. Створити клас, що реалізує динамічний масив як описано в теоретичних відомостях з методами: **pushBack**, **display**, **elementAt**, **getSize**, **getCapacity**, **sort**.
2. Метод **sort** реалізувати за допомогою алгоритму бульбашки.

Рівень D

1. Створити клас, що реалізує динамічний масив з усіма вказаними в теоретичних відомостях методами.

Рівень C

1. Реалізувати повний інтерфейс динамічного масиву та списку.

Рівень B

1. В якості алгоритму сортування динамічного масиву використати алгоритм швидкого сортування (**quick sort**).
2. В якості алгоритму сортування списку використати алгоритм злиття (**merge sort**).
3. Оцінити асимптотичну складність виконання обох алгоритмів.

Рівень A

1. Створити базовий інтерфейс для обох класів, та наслідувати класи від цього загального інтерфейсу.
2. Винести функції **toString** та **display** поза класи таким чином, щоб вони приймали як клас масиву, так і клас списку в якості аргументу.
3. Реалізувати обидва методи сортування для обох структур даних, а також створити функцію, що виконує експеримент з порівняння швидкості сортування обома алгоритмами на обох структурах даних. Зробити висновки.

Рівень GOD*

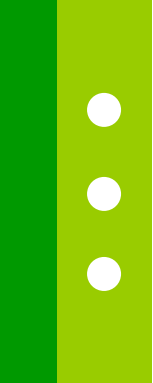
1. Створити клас, що реалізує хеш-таблицю, де ключами та значеннями є текстові рядки. Хеш-функцію обрати довільно.
2. Реалізувати список переповнення як метод розв'язання колізій хеш-функції.
3. Реалізувати методи, що повертають списки ключів та значень хеш-функції.

Контрольні запитання

1. Що таке об'єктно-орієнтоване програмування (ООП)?
2. Три принципи ООП
3. Що таке алгоритм, та які його основні властивості?
4. Що таке обчислювальна складність алгоритму, та яка методика її визначення?
5. Структура даних «динамічний масив».
6. Структура даних «однозв'язний список».
7. Що таке класи та екземпляри класів?
8. Структура класу: члени-дані (поля) та члени-функції (методи).
9. Створення та знищення екземпляру класу: ключові слова **new** та **delete**.
10. Конструктор по замовчуванню.
11. Конструктори копіювання та конструктори копіювання по замовчуванню.
12. Робота з текстовими рядками в C++ - клас **std::string**. Різниця між текстовими рядками в C та C++.

Комп'ютерний практикум №2

Створення графічних додатків

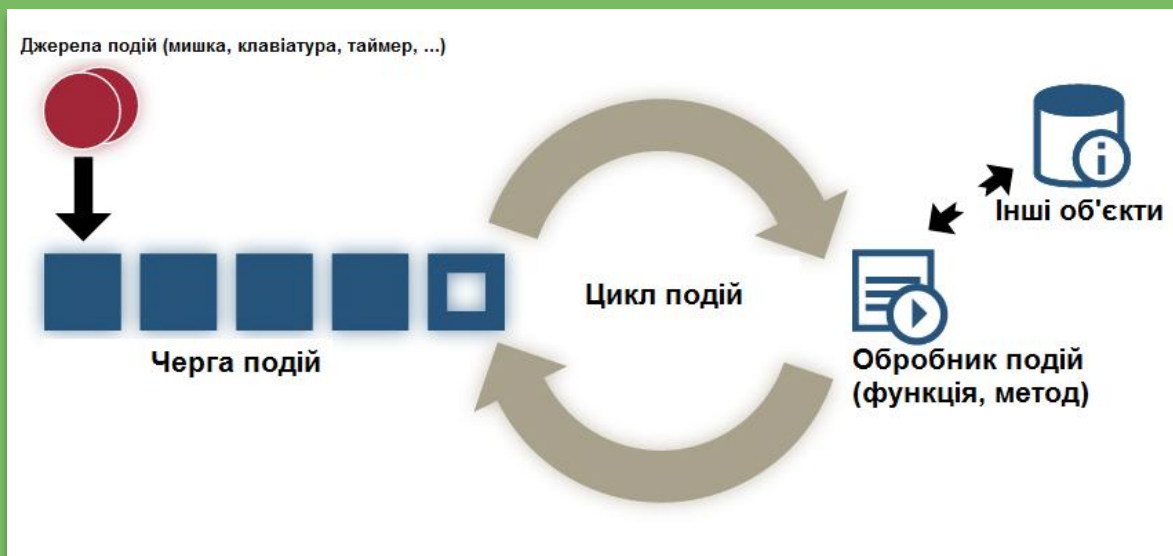
- 
- *Архітектура графічних додатків на прикладі Qt*
 - *Шаблони проектування OO програм*
 - *Шаблон проектування «Singleton»*

Короткі теоретичні відомості

Архітектура графічних додатків

Архітектура графічних додатків базується на так званому **циклі подій (event loop)**. На найвищому рівні абстракції такий додаток являє собою вічний цикл, який на кожній ітерації обробляє вхідні дані програми (події), викликаючи відповідні підпрограми, що реагують на такі події (рис. 1).

Рисунок 2.1. Архітектура графічних додатків (подійно-орієнтована)



Таким чином потік виконання графічної програми виконується наступним чином:

1. Користувач шляхом використання пристрою вводу, або програмний компонент, генерують подію.
2. Подія переноситься в чергу на обробку.
3. Цикл подій вибирає подію з черги подій і викликає обробник подій (зазвичай метод, або функцію), а сама подія та її параметри передаються методу у вигляді параметра.
4. Програмний код методу виконує певні дії впливаючи на інші об'єкти програми.

Розглянемо процес виконання графічної програми на конкретному прикладі:

1. Користувач клацає мишкою на кнопці закриття програми у вікні.
2. Подія натиснення кнопки мишки (**click**) передається у вигляді події в чергу подій. Сама подія являє собою об'єкт, що містить властивості події,

в даному випадку координати мишки на екрані, кнопку мишки, що була натиснута (ліва, права, коліщатко, ...), і т.д..

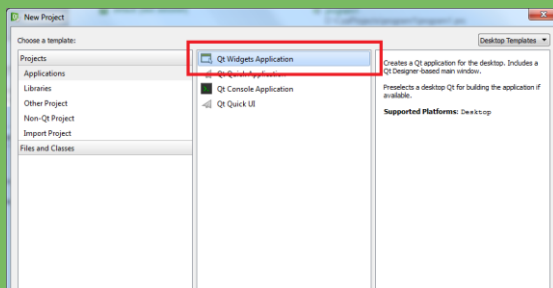
3. Цикл подій обирає цю подію з черги і викликає метод *onClick(QMouseEvent* event)*.
4. Функція *onClick* виконує завершення програми, виконуючи інші необхідні дії.

Мінімальна графічна програма на Qt

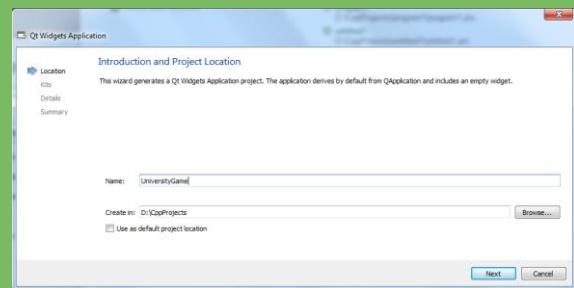
В процесі виконання комп'ютерних практикумів використовується фреймворк Qt. Нижче в прикладі представлена мінімальна графічна програма з використанням даного фреймворку.

На рисунку 2.2 показаний покроковий процес створення мінімального додатку, з якого слід розпочати роботу над подальшими практикумами.

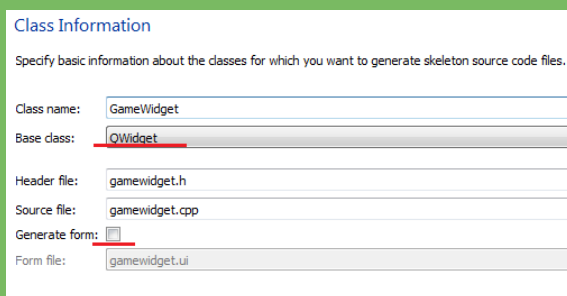
Рисунок 2.2. Покрокова інструкція до створення мінімального Qt додатку



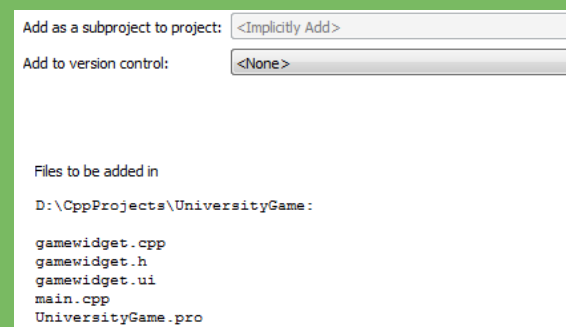
(1)



(2)



(3)



(4)

Для кращого розуміння загальної структури проектів, що створюються за допомогою фреймворку Qt рекомендується пройти декілька наступних уроків з офіційного сайту qt-project.org:

1. [Перша програма на Qt](#)
2. [Створення красивої кнопки](#)
3. [Вступ до сигналів та слотів](#)

Шаблони проектування ПЗ

Шаблони проектування об'єктно-орієнтованих (*Object-Oriented Design Patterns*) програм являють собою усталені стандартні рішення типових задач проектування додатків. Шаблони проектування діляться на 3 групи:

1. **Твірні** (*creational*) – визначають способи створення екземплярів класів.
 - a. **Одинак** (*singleton*) – не дозволяє створювати більше одного екземпляра певного класу.
 - b. **Прототип** (*prototype*) – кожен екземпляр класу створюється шляхом клонування іншого екземпляру класу.
 - c. **Пул об'єктів** (*object pool*) – екземпляри класу створюються і ініціалізуються в певній кількості завчасно, а при запиті на створення нового екземпляру – вибираються і віддаються з колекції (пулу).
 - d. **Фабричний метод** (*factory method*) – надає підкласам певний уніфікований метод для створення конкретних екземплярів класів.
 - e. **Будівельник** (*builder*) – відділяє процес створення складного об'єкта від його представлення таким чином, що в результаті створення об'єкта можна отримати різні представлення.
 - f. **Абстрактна фабрика** (*abstract factory*) – надає єдиний інтерфейс для об'єктів (фабрик), що служать для створення сімейств пов'язаних об'єктів
2. **Структурні** (*structural*) – визначають способи організації відношень між класами в додатку.
 - a. **Адаптер** (*adapter*) – дозволяє використовувати вже створені об'єкти, які вирішують необхідні задачі, але не володіють потрібним клієнтові інтерфейсом.
 - b. **Міст** (*bridge*) – відділяє абстракцію від реалізації таким чином, щоб і абстракція і реалізація могли змінюватись незалежно одна від одної.
 - c. **Компонувальник** (*composite*) – визначає ієрархію класів, що можуть одночасно складатись із примітивних та складних об'єктів, спрощує додавання нових об'єктів до ієрархії.
 - d. **Декоратор** (*decorator*) – дозволяє динамічно підключати до об'єктів певну додаткову поведінку.

- e. **Фасад** (*facade*) – приховує реальну структуру системи, виводячи її взаємодію з іншими об'єктами через один об'єкт, який не виконує корисної роботи, а лише делегує виконання функцій внутрішнім компонентам системи.
 - f. **Пристосуваець** (*flyweight*) – вирішує проблему створення великої кількості маленьких об'єктів зі спільними рисами.
 - g. **Замісник** (*proxy*) – перехоплює та контролює всі виклики до об'єкта, який він заміщує (наприклад, для протоколювання доступу).
3. **Поведінкові** (*behavioral*) – визначають способи взаємодії між об'єктами для вирішення певної задачі.
- a. **Ланцюжок обов'язків** (*chain of responsibility*) – виконує обробку одного і того ж повідомлення в різних компонентах системи.
 - b. **Команда** (*command*) – розділяє два об'єкта таким чином, щоб один об'єкт не залежав від іншого напряму, а такий зв'язок був опосередкованим через об'єкт-команду.
 - c. **Інтерпретатор** (*interpreter*) – виконує вирішення задачі, яка часто змінюється.
 - d. **Ітератор** (*iterator*) – виконує послідовний доступ до елементів контейнера не володіючи інформацією про особливості організації елементів в контейнері.
 - e. **Медіатор** (*mediator*) – організовує взаємодію між декількома об'єктами таким чином, щоб ці об'єкти не звертались один до одного напряму.
 - f. **Хранитель** (*memento*) – дозволяє зафіксувати внутрішній стан об'єкта таким чином, щоб пізніше можна було його відновити.
 - g. **Спостерігач** (*observer*) – надає певному об'єкту (*observable*) механізм, що дозволяє сповіщати інші об'єкти (*observers*) про зміну свого стану.
 - h. **Стан** (*state*) – використовується тоді, коли об'єкт повинен змінювати свою поведінку в залежності від свого стану.
 - i. **Стратегія** (*strategy*) – визначає сімейство взаємозамінних алгоритмів, а також забезпечує їх інкапсуляцію.
 - j. **Шаблонний метод** (*template method*) – дозволяє визначати загальну структуру алгоритму, дозволяючи дочірнім класам змінювати деякі кроки алгоритму, не порушуючи його структуру в цілому.
 - k. **Відвідувач** (*visitor*) – визначає операцію, що виконується над іншими об'єктами, проте не змінює самі ці об'єкти.

Singleton

Шаблон **singleton** використовується у випадку, коли необхідно забезпечити створення не більш, ніж одного екземпляру певного класу. Даний шаблон також забезпечує єдину глобальну точку доступу до даного екземпляру класу,

уникаючи, таким чином необхідності в створенні одної чи більше глобальних змінних для програми.

Ще однією важливою властивістю, якою повинен володіти singleton є потокова безпечність, тобто, навіть при одночасному доступу до об'єкта декількох потоків виконання програми, повинен створюватись лише один екземпляр класу, а також не повинно виникати виключних ситуацій.

Починаючи з версії C++11, найбільш прийнятним вважається реалізація даного шаблону у вигляді так званого singleton'а Мейєрса:

Лістинг 2.1. Singleton Мейєрса.

```
class OnlyOne
{
public:
    static const OnlyOne& Instance()
    {
        static OnlyOne theSingleInstance;
        return theSingleInstance;
    }
private:
    OnlyOne(){};
    OnlyOne(const OnlyOne& root);
    OnlyOne& operator=(const OnlyOne&);
};
```

Завдання до виконання

Ціллю даного і наступних практикумів в рамках даного курсу є створення комп'ютерної гри з використанням мови програмування C++11 та фреймворку Qt 5.4.

1. Створити мінімальний проект Qt додатку згідно з інструкціями поданими в теоретичних відомостях.
2. Створити **singleton** клас **Game** (можна в довільному варіанті реалізації синглтона), що буде містити інформацію про загальний стан гри. Цей клас повинен мати публічний метод **run**, що виконує підготовку та показ **GameWidget**, конструювання якого можна винести в конструктор класу **Game**.
3. В проекті вже виконано наслідування класу **GameWidget** від класу **QWidget**. Виконати перевизначення методу **paintEvent**. Для підказки

можна скористатись [прикладом додатку](#), таким чином, щоб виконати прорисовку екран завантаження вашої гри (конкретна гра - в залежності від варіанту в таблиці нижче).

4. В екрані завантаження повинно бути виконане промальовування як статичної графіки (засобами [QImage](#), [QPixmap](#), [QPainter](#)), так і анімації (за допомогою [QTimer](#)).
5. Всю графіку, а також додаткові файли, наприклад, файли для організації карт гри зберігати і завантажувати з [файлів ресурсів qrc](#).

Варіанти

1. [Класичний арканойд](#)
 - a. Дошка рухається по горизонталі (управляється клавішами)
 - b. М'ячик відбивається від стін і збиває цеглинки
 - c. Всі цеглинки однакові, знищуються при дотику до м'ячика.
 - d. Гра закінчується, якщо м'ячик торкається нижнього краю екрану.
2. [Тетріс](#)
 - a. Класичний тетріс
3. [Танчики](#)
 - a. Є тільки один тип стін, що знищується пострілом
 - b. Бонусів немає, танк не покращується.
 - c. Ворожі танки теж не мають рівнів, стріляють і рухаються хаотично (*random*).
 - d. Бази немає, гра триває, поки ворожий танк не підіб'є танк гравця.
4. [Астероїди](#)
 - a. Астероїди розпадаються на менші, як і в грі за посиланням.
 - b. Астероїди не пролітають межі екрану, а відбиваються від його стінок.
5. [Змійка](#)
 - a. За правилами, як реалізовано за посиланням.
 - b. З кожним новим сегментом швидкість змійки зростає.

! Важливо ! В межах 2-го практикуму ***не потрібно*** реалізовувати повністю ігри, а тільки те, що вказано в кроках. Варіанти потрібні для того, щоб орієнтуватись в завданнях по наступних лабораторних.

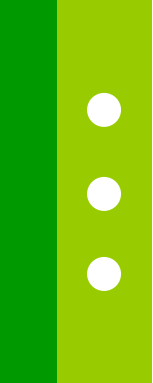
Контрольні запитання

1. Що таке шаблон проектування в ООП? На які типи діляться шаблони проектування?

2. Шаблон singleton? Які вимоги до нього ставляться? Якою є найпопулярніша реалізація singleton для C++11?
3. Що таке конструктор, коли він викликається?
4. Що таке деструктор, коли він викликається?
5. Шаблони проектування Стан та Стратегія.
6. Шаблони проектування Фабричний метод та Ланцюжок обов'язків.
7. Шаблони проектування Ітератор та Адаптер.
8. Шаблони проектування Команда та Інтерпретатор.
9. Шаблони проектування Спостерігач та Декоратор.

Комп'ютерний практикум №3

Проектування головного МЕНЮ

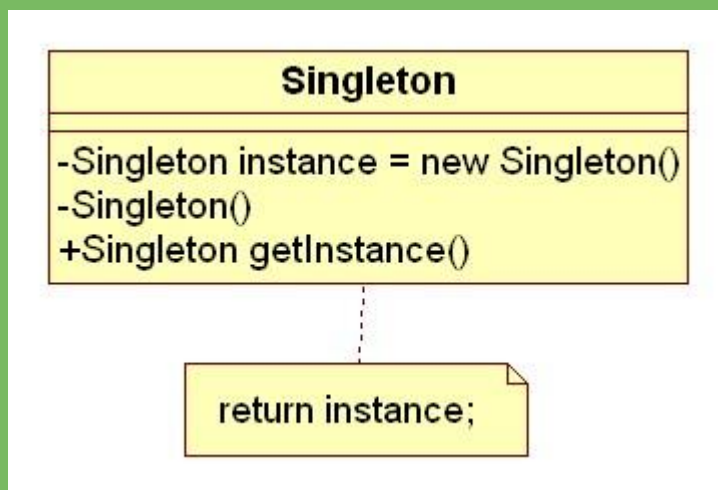
- 
- *Шаблон проектування «Стан»*
 - *Шаблони проектування «Стратегія» та «Стан»*
 - *Організація журналу (log) в додатку*

Короткі теоретичні відомості

Структура об'єктно-орієнтованих програм, на відміну від алгоритмів, не подається блок-схемами. Існують різні методи для їх, ОО програм, структурного подання, проте найбільш поширеним і загальноприйнятим способом є використання уніфікованої мови моделювання [UML \(Unified Modeling Language\)](#). Дана мова є універсальною графічною мовою моделювання структур та процесів, що застосовується в різних галузях знань; для моделювання структури та зв'язків між об'єктами в ОО програмах використовується спеціально створений для цього підрозділ мови UML. Ознайомитись з даною мовою можна за [посиланням](#).

Для прикладу, розглянутий в попередньому практикумі шаблон «Singleton» може бути поданий на мові UML як показано на *діаграмі 3.1*.

Діаграма 3.1. UML - діаграма шаблону «Singleton»



Шаблони проектування «Стан» та «Стратегія»

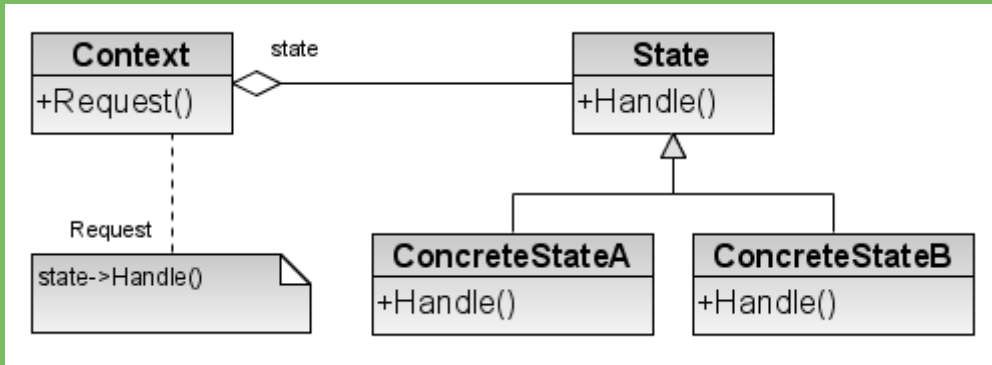
Шаблон проектування «State» - це поведінковий шаблон проектування, що використовується для інкапсуляції різних варіантів поведінки в залежності від певного внутрішнього стану об'єкта.

Застосовується даний шаблон проектування в наступних випадках:

- поведінка об'єкта залежить від його стану та може змінюватись в процесі роботи програми
- потрібно позбутись складних розгалужень в алгоритмі роботи програми, котрі б управляли поведінкою класу

Структурно даний шаблон проектування можна подати у вигляді наступної UML-діаграми:

Діаграма 3.2. UML - діаграма шаблону «State»



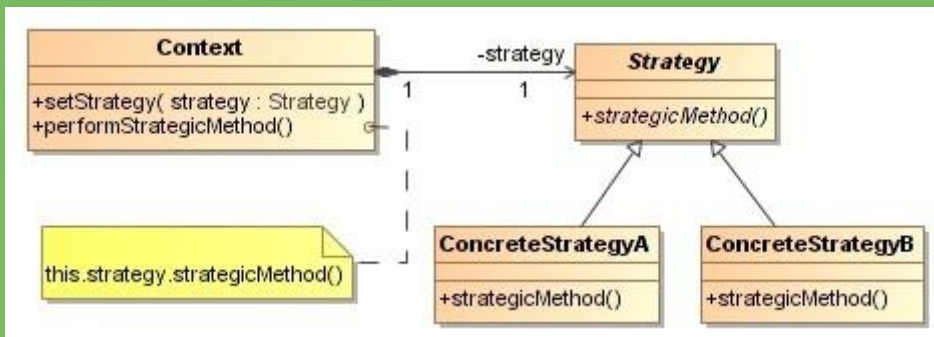
Шаблон проектування «Strategy» - це поведінковий шаблон проектування, що використовується для інкапсуляції різних алгоритмів поведінки об'єкта, проте вибір шаблону поведінки, на відміну від шаблону «State», стратегія поведінки обирається клієнтським кодом, а не кодом самого класу.

Застосовується, коли:

- Програма повинна забезпечувати різні варіанти поведінки чи алгоритму
- Потрібно змінювати поведінку кожного екземпляра класу на етапі виконання

Структурно даний шаблон проектування можна подати у вигляді наступної UML-діаграми:

Діаграма 3.3. UML - діаграма шаблону «Strategy»



Організація журналу додатку

Журналом додатку є, зазвичай, текстовий файл, консоль, чи база даних, що відображає основні віхи роботи програми і використовується для відлагодження додатків. Журнал додатку ведеться як в процесі розробки (debug log), так і на етапі використання додатку кінцевим користувачем (log).

Одним зі способів організації поділу журналу на режим розробки та використання додатку є введення певної глобально доступної булевої константи, в залежності від значення якої певні події записуються до журналу, чи не записуються.

Наприклад, на етапі розробки можна записувати до журналу стани всіх ігрових об'єктів, тоді як на етапі застосування додатку така інформація не є потрібною.

Також часто в журналах додатків використовуються так звані рівні записів журналу, що позначаються цілими константами і визначають рівень подробиць про хід роботи додатку, який записується в журнал.

Одним зі способів організації журналу є використання шаблону «Singleton».

Завдання до виконання

Ціллю даного і наступних практикумів в рамках даного курсу є створення комп'ютерної гри з використанням мови програмування C++11 та фреймворку Qt 5.4.

1. Реалізувати стани комп'ютерної гри, що представляють собою різні екрани гри з використанням шаблону проектування «State» чи «Strategy»:
 - a. Екран вступу / завантаження (після запуску)
 - b. Головне меню (після екрану вступу)
 - c. Екран власне гри (після головного меню)
 - d. Таблиця результатів (після головного меню чи екрану гри)
2. Реалізувати головне меню з наступними пунктами, а також обробку вибору меню з використанням шаблону проектування «Strategy»:
 - a. Нова гра
 - b. Таблиця результатів
 - c. Вихід.
3. Реалізувати журнал в додатку, що відображав би основні віхи роботи програми, стани об'єктів, а також переходи між станами всередині програми.

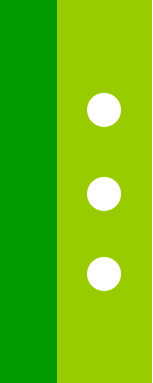
4. Журнал повинен бути доступний при розробці додатку і недоступний в випусковій версії, або доступний у вигляді запису до файлу в короткому вигляді.

Контрольні запитання

1. Шаблон проектування «Стратегія».
2. Шаблон проектування «Стан».
3. Наслідування в C++.
4. Множинне наслідування в C++.
5. Управління доступом до членів класів-предків.
6. Абстрактні класи.
7. Інтерфейси.
8. Віртуальні функції в C++.
9. Віртуальні деструктори.
10. Чисті віртуальні функції в C++.

Комп'ютерний практикум №4

Проектування ігрового процесу

- 
- *Наслідування та поліморфізм*
 - *Шаблон проектування «Фабричний метод»*
 - *Шаблон проектування «Шаблонний метод»*

Короткі теоретичні відомості

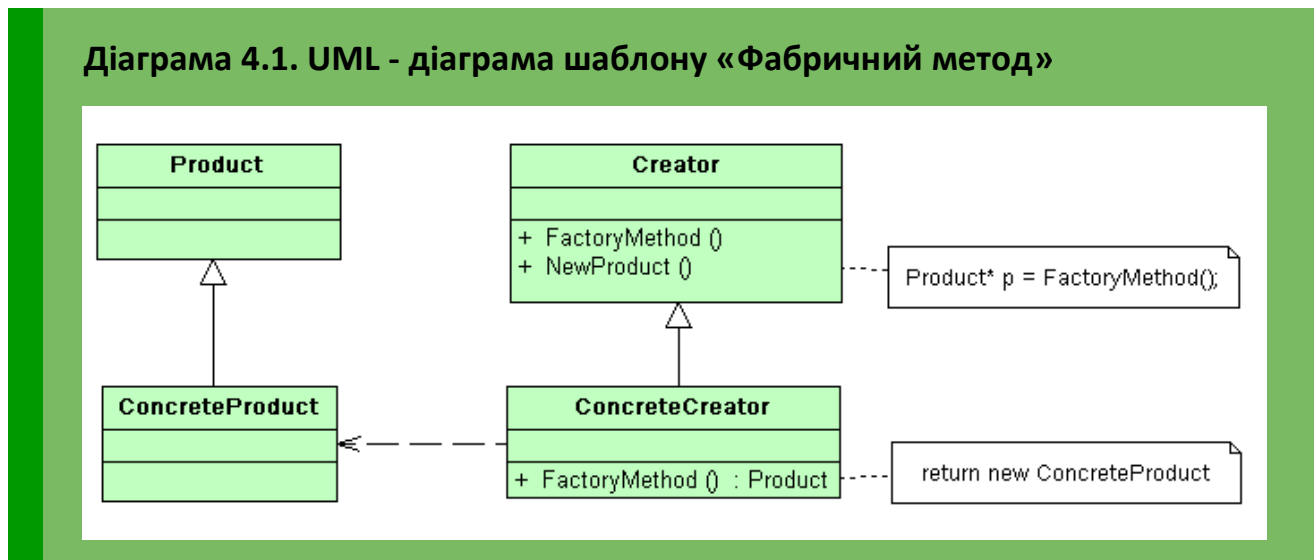
Шаблон проектування «Фабричний метод»

Шаблон проектування «Фабричний метод» - це твірний шаблон проектування, що надає підкласам інтерфейс для створення нових екземплярів деякого класу, тобто створення екземплярів делегується дочірнім класам.

Застосовується даний шаблон проектування в наступних випадках:

- коли завчасно невідомо екземпляри яких саме класів потрібно створювати
- клас спроектований таким чином, щоб об'єкти, які він створює специфікувались підкласами

Структурно даний шаблон проектування можна подати у вигляді наступної UML-діаграми:



Шаблон проектування «Шаблонний метод»

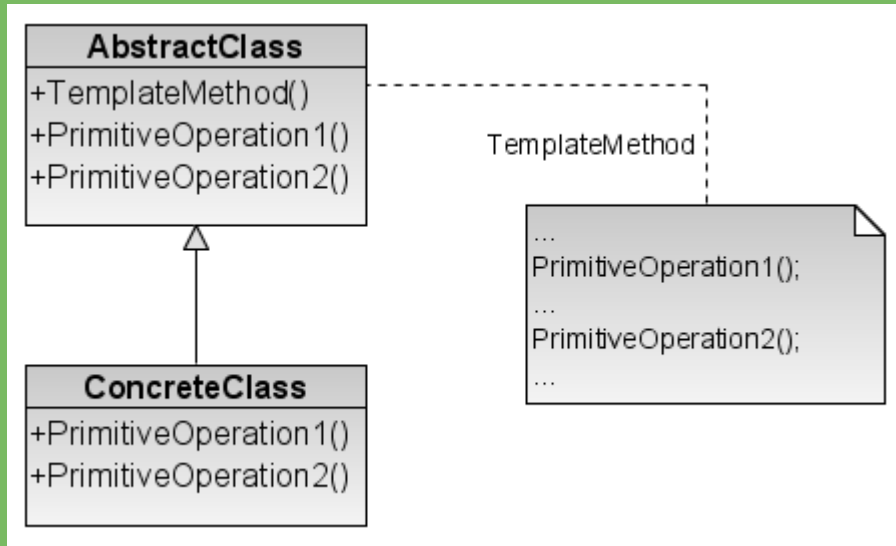
Шаблонний метод – це шаблон проектування, що дозволяє підкласам перевизначати деякі кроки певного алгоритму, не змінюючи структуру алгоритму в цілому. Наприклад, ігровий об'єкт має певні правила поведінки, проте управляється він може як живим гравцем, так і певним алгоритмом, і тільки цей факт встановлює між двома такими об'єктами відмінність.

Використовується даний шаблон проектування, коли:

- треба одноразово задати незалежні етапи певного алгоритму
- треба відокремити в одному класі поведінку, що є загальною для усіх підкласів
- для управління розширенням підкласів

Структурно даний шаблон проектування можна подати у вигляді наступної UML-діаграми:

Діаграма 4.2. UML - діаграма шаблону «Шаблонний метод»



Організація ігрового процесу

Організація ігрового процесу в комп'ютерній грі передбачає створення взаємодіючих об'єктів, стани яких впливають один на одного.

На кожному кроці деякого таймера необхідно змінювати стани цих об'єктів в залежності від впливу гравця чи внутрішньоігрових об'єктів один на одного.

Вищеописані шаблони дозволяють впорядкувати створення та організацію поведінки внутрішньоігрових об'єктів.

Завдання до виконання

Ціллю даного і наступних практикумів в рамках даного курсу є створення комп'ютерної гри з використанням мови програмування C++11 та фреймворку Qt 5.4.

1. Реалізувати ігровий процес в залежності від отриманого варіанту (гри), а саме – статичні та динамічні ігрові об'єкти, штучний інтелект для динамічних об'єктів, а також управління гравцем за допомогою клавіатури чи мишки.

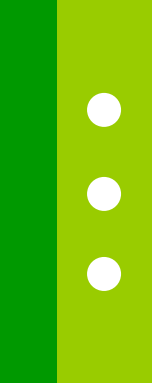
2. Застосовуючи шаблони проектування (як вищеописані, так і інші, що були наведені в Комп'ютерному Практикумі №2, для мінімізації коду, та досягнення якомога більшої гнучкості архітектури програми).
3. Ввести критерії виграшу / програшу.
4. Додати підрахунок ігрових очок для подальшої реалізації і використання в таблиці результатів.

Контрольні запитання

1. Шаблон проектування «Фабричний метод».
2. Шаблон проектування «Шаблонний метод».
3. Шаблон проектування «Посередник».
4. Шаблон проектування «Хранитель».
5. Шаблон проектування «Спостерігач».

Комп'ютерний практикум №5

Проектування таблиці результатів / лідерів

- 
- *Мета програмування*
 - *Шаблони та спеціалізація шаблонів в C++*
 - *STL – стандартна бібліотека шаблонів*

Короткі теоретичні відомості

Standard Templates Library (STL)

Стандартна бібліотека шаблонів являє собою набір стабільних, злагоджених узагальнених компонентів для мови програмування C++.

Бібліотека містить 5 типів компонентів, найпопулярнішими з яких є:

1. **Алгоритм** (algorithm) – визначає загальнозживані алгоритми
 - a. **find** – пошук в контейнері
 - b. **count** – підрахунок об'єктів
 - c. **search**
 - d. **sort** – сортування контейнера
 - e. **merge** – злиття контейнерів
 - f. **min** – пошук мінімального елемента
 - g. **max** – пошук максимального елемента
2. **Контейнер** (container) – визначає популярні набори об'єктів в пам'яті
 - a. **vector** – динамічний масив
 - b. **deque** – двостороння черга
 - c. **list** – список
 - d. **set** – множина
 - e. **map** – контейнер типу «ключ-значення»
 - f. **hash_map** – хешований контейнер типу «ключ-значення»
3. **Ітератор** (iterator) – забезпечує для алгоритму засоби доступу до вмісту контейнера
4. **Функціональний об'єкт** (*functional object*) – інкапсулює функцію в об'єкті для використання іншими компонентами.
5. **Адаптер** (*adaptor*) – адаптує компонент для забезпечення іншого інтерфейсу.

Завдання до виконання

Ціллю даного і наступних практикумів в рамках даного курсу є створення комп'ютерної гри з використанням мови програмування C++11 та фреймворку Qt 5.4.

1. Реалізувати таблицю результатів гри (переможців).
2. Таблиця результатів повинна бути відсортована по спаданню набраних очок.
3. Таблиця результатів повинна бути відображення після закінчення гри, а також доступна з головного меню програми.
4. Сортування таблиці результатів реалізувати у вигляді C++ шаблону (не використовуючи STL sort).

5. Переглянути програмний код, створений протягом попередніх практикумів, замінити, де це необхідно, створені вручну алгоритми та структури даних на ті, що доступні в рамках STL (крім сортування).

Контрольні запитання

1. Призначення і задачі STL.
2. Типи компонентів STL.
3. Алгоритми сортування.
4. Швидке сортування (quicksort).
5. Шаблони в C++.
6. Спеціалізація шаблонів в C++.