Практичні роботи з курсу «Програмування мовою Java»  
Зміст

[Практична робота №1 Структура проекту 2](#_Toc22824002)

[Основні відомості про проект 2](#_Toc22824003)

[Об’єкти «Задача» 2](#_Toc22824004)

[Завдання. Частина 1 3](#_Toc22824005)

[Відомості щодо виконання роботи 3](#_Toc22824006)

[Завдання. Частина 2 3](#_Toc22824007)

[Практична робота 2 Масиви та посилання 5](#_Toc22824008)

[Завдання. Частина 1 5](#_Toc22824009)

[Завдання. Частина 2 5](#_Toc22824010)

[Практична робота 3 Виключні ситуації 6](#_Toc22824011)

[Завдання 1. Виключні ситуації 6](#_Toc22824012)

[Завдання 2. Робота зі зв’язним списком 6](#_Toc22824013)

[Практична робота 4 Спадкування 7](#_Toc22824014)

[Завдання 1. Спадкування 7](#_Toc22824015)

[Завдання 2. Абстрактна фабрика 7](#_Toc22824016)

[Практична робота 5 Ітератори, сервісні методи 8](#_Toc22824017)

[Завдання 1. Ітератори 8](#_Toc22824018)

[Завдання 2. Сервісні методи 8](#_Toc22824019)

[Практична робота 6 Java 8 9](#_Toc22824020)

[Завдання 1.Використання потоків 9](#_Toc22824021)

[Практична робота 7 Сервісні класи та колекції 10](#_Toc22824022)

[Завдання 1. Перехід до використання класу LocalDateTime 10](#_Toc22824023)

[Завдання 2. Виділення логіки роботи з календарем 10](#_Toc22824024)

[Практична робота 8 Ввід-вивід та серіалізація 11](#_Toc22824025)

[Завдання 1. Серіалізація 11](#_Toc22824026)

[Завдання 2. Бінарний ввід-вивід 11](#_Toc22824027)

[Завдання 3. Текстовий ввід-вивід 12](#_Toc22824028)

[Комплексна практична робота 9 Повноцінний додаток «Task Manager» 13](#_Toc22824029)

[Загальні відомості 13](#_Toc22824030)

[Основні вимоги 13](#_Toc22824031)

[Дизайн додатку 13](#_Toc22824032)

[Communication Diagram 13](#_Toc22824033)

[Class Diagram 14](#_Toc22824034)

# Практична робота №1 Структура проекту

## Основні відомості про проект

Практичні роботи першого семестру являють собою розвиток єдиного проекту, його удосконалення від простого класу до повноцінного користувацького додатку. Темою роботи є задачі, які може поставити собі користувач, наприклад «Піти з друзями у боулінг в середу», або «Бігати по 3 км кожного дня о шостій годині». Проект, що є темою цих практичних робіт є додатком, що допоможе користувачу зберігати, переглядати задачі та не забувати їх виконувати.

## Об’єкти «Задача»

Основними об’єктами, з якими буде працювати додаток, є задачі.

Задачі мають деякий текст, що описує *деталі задачі*, наприклад «Прибирання в кімнаті».

Крім того задачі можуть бути *активними та неактивними* – наприклад на час відпустки задача «Ранкова пробіжка» може бути неактивною і тимчасово не виконуватись.

Для опису часу для початку будуть використовуватись цілі числа, що означають наприклад кількість годин, що пройшли з початку відліку часу (наприклад з 1-го лютого 2000 року 00:00), так число 36 буде означати 12:00 2-го лютого 2000 року.

Задачі можуть бути заплановані на виконання *один раз*, наприклад «Зустріч у кафе 26 червня о 18:00»:

Час

«Зустріч у кафе»

0

26 червня

Або задача може бути запланована на *регулярне виконання* впродовж деякого проміжку часу з заданим інтервалом (у годинах), наприклад «Ранкова пробіжка з 1 червня по 5 червня кожну добу о 8:00»:

Час

«Ранкова пробіжка»

0

Початок о 8:00   
1-го червня

Закінчення о 00:00   
5-го червня

Інтервал повторення 24 години

Таким чином метою першої практичної роботи буде створити клас об’єктів Задача.

## Завдання. Частина 1

Створіть клас Task у пакеті ua.sumdu.j2se.studentName.tasks (замініть studentName на ваше ім’я) із наступними публічними методами:

* Конструктор Task(String title, int time), що конструює неактивну задачу, яка виконується у заданий час без повторення із заданою назвою.
* Конструктор Task(String title, int start, int end, int interval), що конструює неактивну задачу, яка виконується у заданому проміжку часу (і початок і кінець включно) із заданим інтервалом і має задану назву.
* Методи для зчитування та встановлення назви задачі: String getTitle(), void setTitle(String title).
* Методи для зчитування та встановлення стану задачі: boolean isActive(), void setActive(boolean active).
* Методи для зчитування та зміни часу виконання для задач, що не повторюються:
  + int getTime(), у разі, якщо задача повторюється метод має повертати час початку повторення;
  + void setTime(int time), у разі, якщо задача повторювалась, вона має стати такою, що не повторюється.
* Методи для зчитування та зміни часу виконання для задач, що повторюються:
  + int getStartTime(), у разі, якщо задача не повторюється метод має повертати час виконання задачі;
  + int getEndTime(), у разі, якщо задача не повторюється метод має повертати час виконання задачі;
  + int getRepeatInterval(), у разі, якщо задача не повторюється метод має повертати 0;
  + void setTime(int start, int end, int interval), у разі, якщо задача не повторювалася метод має стати такою, що повторюється.
* Метод для перевірки повторюваності задачі boolean isRepeated().

Відкомпілюйте проект використовуючи меню *Macro > Зібрати Java Проект*, проект має збиратися без помилок компіляції, та бажано без помилок оформлення.

## Відомості щодо виконання роботи

При виконанні роботи необхідно оформлювати файли класів відповідно до Java Code Conventions, з якими можна ознайомитися за адресою <http://www.oracle.com/technetwork/java/codeconv-138413.html>. Крім того всі публічні елементи повинні мати javadoc, що коротко пояснює призначення і використання елементу. При написанні такої документації не потрібно описувати речі, які легко зрозуміти із декларації, наприклад типи параметрів і результатів, необхідно зазначати неочевидні речі – обмеження на значення параметрів, спеціальні результуючі значення, можливі виключні ситуації.

## Завдання. Частина 2

Важливою операцією при роботі з задачами є знаходження наступного моменту виконання задачі.

Так, якщо задача не є активною, то вона не виконується ніколи, якщо задача активна і виконується лише один раз, то наступним моментом виконання буде або цей єдиний момент, або ніколи (якщо задача вже була виконана):

Час

«Зустріч у кафе»

0

26 червня

Відносно даного моменту   
наступний – 26 червня

Відносно даного моменту   
задача більше не виконується

23 червня

27 червня

Якщо задача є активною і повторюється, то аналогічно знаходиться наступний момент виконання задачі відносно даного часу:

Час

«Ранкова пробіжка»

0

Початок   
1-го червня

Закінчення   
31-го серпня

Інтервал повторення 24 години

Відносно даних моментів наступні помічені

Відносно даного моменту   
задача більше не виконується

Необхідно додати у класс Task метод int nextTimeAfter(int current), що повертає час наступного виконання задачі після вказаного часу current, якщо після вказаного часу задача не виконується, то метод має повертати -1.

# Практична робота 2 Масиви та посилання

Для роботи із задачами у попередній практиці був розроблений клас об’єктів Task, темою другої практичної роботи є розробка списку задач, що дозволяє працювати відразу з декількома задачами. Задачі у списку можуть повторюватись, порядок слідування задач не має значення, але не повинен змінюватися, якщо у список не додавалися та не видалялися з нього задачі.

## Завдання. Частина 1

Створіть клас ArrayTaskList у пакеті ua.sumdu.j2se.studentName.tasks (замініть studentName на ваше ім’я) із наступними публічними методами:

* void add(Task task) – метод, що додає до списку вказану задачу.
* boolean remove(Task task) – метод, що видаляє задачу із списку і повертає істину, якщо така задача була у списку. Якщо у списку було декілька таких задач, необхідно видалити одну будь-яку.
* int size() – метод, що повертає кількість задач у списку.
* Task getTask(int index) – метод, що повертає задачу, яка знаходиться на вказаному місці у списку, перша задача має індекс 0.

Задачі у списку повинні зберігатися за допомогою масиву, список може містити будь-яку кількість задач, що може знаходитись у масиві, але не повинен виділяти набагато більше місця, ніж потрібно у даний момент. Наприклад, якщо у списку п’ять задач, то масив для їх зберігання не повинен бути розміром у 100 задач.

## Завдання. Частина 2

Крім того для списку задач необхідно знаходити, які саме задачі будуть виконані хоча б раз у деякому проміжку, наприклад які задачі заплановані на наступний тиждень. Для цього створіть у класі ArrayTaskList метод ArrayTaskList incoming(int from, int to) – метод, що повертає підмножину задач, які заплановані на виконання хоча б раз після часу from і не пізніше ніж to.

**Наприклад**, нехай у нашому списку знаходяться наступні задачі:

1. Обід із гарною дівчиною 24 серпня о 16:00.
2. Ранкова пробіжка з 1 березня по 1 вересня кожну добу.
3. Прийом ліків з 20 серпня по 28 серпня кожні 12 годин.
4. Зустріч з друзями 1 вересня о 18:00.

Тоді задачі, що відбудуться з 25 серпня 8:00 по 26 серпня 8:00 будуть «*Ранкова пробіжка*» та «*Прийом ліків*».

# Практична робота 3 Виключні ситуації

## Завдання 1. Виключні ситуації

До цього моменту у завданнях не фігурувало, що має відбуватися, якщо виконуються недопустимі операції, наприклад, якщо зі списку із трьох задач намагаються отримати задачу під номером 5.

Вам необхідно проаналізувати існуючий код і вирішити, що робити у подібних ситуаціях, майте на увазі що:

1. У попередніх завданнях час задавався цілим числом, як кількість одиниць часу, що пройшли із деякого початку відліку. Згідно з цим відмітки часу не можуть бути від’ємними.
2. Список задач має містити задачі, тому у нього не можна додавати порожні посилання (null).
3. Інтервал повторення задачі повинен бути більше нуля.

Конструктор класу Task обов’язково повинен генерувати включення IllegalArgumentException у випадку, коли час було передано від’ємним числом, а метод getTask повинен генерувати IndexOutOfBoundsException у випадку, коли індекс виходить за допустимі межі для списку.

В усіх інших випадках рішення щодо того, що саме робити у випадках порушення обмежень і якщо породжувати виключення певного типу, повинні бути аргументовані.

## Завдання 2. Робота зі зв’язним списком

Концепція списку задач не залежить від способу зберігання задач, користувачі об’єктів класу ArrayTaskList можуть навіть не знати, як саме цей клас реалізований. Однак реалізація через масив має свої недоліки – повільна операція видалення задачі. Тому для сценаріїв, коли видалення задач відбувається часто необхідно створити список задач, що буде зберігати задачі у зв’язному списку (однозв’язний, двозв’язний, або інша модифікація на вибір, не можна використовувати вже існуючі реалізації, такі як java.util.LinkedList), який не має цього недоліку. Створіть клас LinkedTaskList у тому ж пакеті із такими самими методами, що і ArrayTaskList (у методі incoming змінить тип об’єкту, що повертається на LinkedTaskList). Об’єкти цього класу мають поводити себе так само, як і об’єкти класу ArrayTaskList.

# 

# Практична робота 4 Спадкування

## Завдання 1. Спадкування

Призначення методів у класах ArrayTaskList і LinkedTaskList майже однакове, що дозволяє написати програму більш абстрактного рівня.

* Оскільки обидва класи ArrayTaskList і LinkedTaskList реалізують один і той самий тип даних, необхідно створити абстрактний клас AbstractTaskList, де описати методи, характерні для списку задач як абстрактні та успадкувати обидва класи списків від цього класу.
* Оскільки реалізація методу incoming не залежить від способу зберігання задач, його можна реалізувати у абстрактному класі AbstractTaskList (при цьому тип об’єкту, що ним повертається також зміниться на AbstractTaskList), таким чином зменшивши дублювання коду.

Таким чином ми отримаємо:

* абстрактний клас AbstractTaskList, що описує операції, які можна виконувати із списком задач, та реалізує методи, що не залежать від способу зберігання;
* клас ArrayTaskList, що успадкований від TaskList та реалізує ті абстрактні операції TaskList, що залежать від способу зберігання. використовуючи масив;
* клас LinkedTaskList, що успадкований від TaskList та реалізує ті абстрактні операції TaskList, що залежать від способу зберігання, використовуючи зв’язний список.

## Завдання 2. Абстрактна фабрика

На даному етапі ми маємо два класи ArrayTaskList і LinkedTaskList, і спільний для них батьківський клас AbstractTaskList. Було б досить зручно мати окремий клас, який би мав метод, що створює об’єкт ArrayTaskList і LinkedTaskList в залежності від параметру, що в нього передається. Це і покладено в основу патерну **абстрактна фабрика**.

* Створіть клас ListTypes з полем types типу enum. types повинно містити значення ARRAY та LINKED.
* Створіть клас TaskListFactory зі статичним методом
  + TaskList createTaskList(ListTypes.types type). Даний метод, відповідно до параметру type, повинен повертати об'єкт класу ArrayTaskList або LinkedTaskList.

# Практична робота 5 Ітератори, сервісні методи

## Завдання 1. Ітератори

Для абстрактного опису об’єктів, що складаються із послідовності об’єктів іншого типу у стандартній бібліотеці існує інтерфейс java.util.Iterable<T>, де T це тип об’єктів – складових частин. Оскільки клас TaskList описує колекцію об’єктів-задач, то він має імплементувати цей інтерфейс.

При цьому інтерфейс Iterable<T> описує абстрактну ітерацію по набору об’єктів, реалізація ж цього інтерфейсу має бути оптимальною с точки зору внутрішньої структури набору. Так, наприклад, LinkedTaskList не повинен починати шукати елемент із самого початку при переході від поточного елементу до наступного.

Також треба мати на увазі, що у одного списку задач може бути одночасно декілька незалежних ітераторів.

## Завдання 2. Сервісні методи

Імплементувати сервісні методи із класу Object:

1. Методи equals та hashCode у всіх класах. Списки задач вважаються однаковими, якщо містять однакові задачі в тому ж самому порядку. Задачі вважаються однаковими, якщо їх властивості рівні.
2. Метод toString для всіх класів. Рядкове відображення повинно включати максимум інформації у зручному вигляді.
3. Задачі і списки задач повинні дозволяти клонувати себе. Реалізація не повинна залучати виклик конструктора класу.

# 

# Практична робота 6 Java 8

## Завдання 1.Використання потоків

Java 8 принесла додаткові можливості для роботи з колекціями з використанням функціонального програмування.

1. Необхідно додати до класу AbstractTaskList метод Stream<Task> getStream() , призначення якого полягає в тому, щоб можна було працювати з колекціями як з
2. Даний метод повинен бути перевизначений у дочірних класах.
3. Перенести реалізацію методу incoming(int from, int to) із класів ArrayTaskList і LinkedTaskList в AbstractTaskList. Тобто у класах ArrayTaskList і LinkedTaskList не має бути методу incoming(int from, int to).
4. Описати логіку методу incoming(int from, int to) в AbstractTaskList з використанням stream API, що з'явився в Java 8.
5. Заборонити методу incoming(int from, int to) бути перевизначеним у дочірних класах, він повинен мати реалізацію лише у класі AbstractTaskList.

# Практична робота 7 Сервісні класи та колекції

## Завдання 1. Перехід до використання класу LocalDateTime

До цього моменту для роботи із моментами часу використовувалися цілі числа, однак у стандартній бібліотеці існує клас java.time.LocalDateTime для представлення дати та часу. Необхідно замінити використання цілих чисел на LocalDateTime для роботи із моментами часу.

При цьому необхідно мати на увазі, що об’єкти класу LocalDateTime на відміну від цілих чисел можуть змінюватись (якщо у них визивається метод setTime), тому такі об’єкти краще копіювати, зберігаючи тільки локальну копію.

Для порівнянь дат необхідно використовувати методи класу LocalDateTime замість приведення дати до цілого числа.

## Завдання 2. Виділення логіки роботи з календарем

На даний момент класи списків задач містять одночасно і логіку зберігання задач і логіку роботи із часом виконання задач (метод incoming).

Для того, щоб дозволити використання будь-якої колекції для зберігання задач необхідно створити окремий клас для роботи із колекціями задач – Tasks і перенести у нього метод incoming у вигляді статичного методу. При цьому метод необхідно абстрагувати від зписків задач – сигнатура методу тепер буде Iterable<Task> incoming(Iterable<Task> tasks, LocalDateTime start, LocalDateTime end).

Окрім того у класі Tasks необхідно реалізувати ще один статичний метод SortedMap<Date, Set<Task>> calendar(Iterable<Task> tasks, LocalDateTime start, LocalDateTime end), який буде будувати календар задач на заданий період – таблицю, де кожній даті відповідає множина задач, що мають бути виконані в цей час, при чому одна задача може зустрічатись відповідно до декількох дат, якщо вона має бути виконана декілька разів за вказаний період.

**Наприклад**, нехай у нашому списку знаходяться наступні задачі:

1. Обід із гарною дівчиною 24 серпня о 16:00.
2. Ранкова пробіжка з 1 березня по 1 вересня кожну добу о 8:15.
3. Прийом ліків з 20 серпня 8:15 по 28 серпня кожні 12 годин.
4. Зустріч з друзями 1 вересня о 18:00.

Тоді календар задач, що відбудуться з 25 серпня 8:00 по 26 серпня 8:00 будуть:

|  |  |
| --- | --- |
| Дата | Задачі |
| 25 серпня 8:15 | Ранкова пробіжка, Прийом ліків |
| 25 серпня 20:15 | Прийом ліків |

При виконанні роботи необхідно мати на увазі, що всі класи колекцій у java.util можуть змінювати свій вміст після створення, так наприклад при заповненні календаря Set, що поміщається у Map не обов’язково має містити одразу всі задачі, їх можна додати до множини пізніше.

# Практична робота 8 Ввід-вивід та серіалізація

Для реалізації повноцінної програми по роботі з задачами необхідно дозволити зберігати задачі і списки задач на диску та передавати по мережі. Для цього необхідно реалізувати методи запису та зчитування списків задач у різних форматах.

## Завдання 1. Серіалізація

Класи-реалізації TaskList та Task необхідно зробити такими, що можна серіалізувати стандартними засобами.

## Завдання 2. Бінарний ввід-вивід

Необхідно створити клас TaskIO із статичними методами:

* void write(TaskList tasks, OutputStream out) – записує задачі із списку у потік у бінарному форматі, описаному нижче.
* void read(TaskList tasks, InputStream in) – зчитує задачі із потоку у даний список задач.
* void writeBinary(TaskList tasks, File file) – записує задачі із списку у файл.
* void readBinary(TaskList tasks, File file) – зчитує задачі із файлу у список задач.

Формат представлення списку задач у двійковому вигляді:

Кількість задач

Довжина назви

Назва

Активність: 0/1

Інтервал повторення

Час початку

Час виконання

Час закінчення

Якщо повторюється

Якщо **не** повторюється

× кількість задач

Для роботи із двійковими потоками зручно використовувати типи DataInput та DataOutput. Дати записувати і зчитувати переводячи у цілі числа.

## Завдання 3. Текстовий ввід-вивід

У класі TaskIO необхідно описати статичні методи:

* void write(TaskList tasks, Writer out) – записує задачі зі списку у потік в форматі JSON.
* void read(TaskList tasks, Reader in) – зчитує задачі із потоку у список.
* void writeText(TaskList tasks, File file) – записує задачі у файл у форматі JSON
* void readText(TaskList tasks, File file) – зчитує задачі із файлу.

Для роботи з форматом JSON рекомендується використовувати бібліотеку Gson. Дану бібліотеку можна підключити використовуючи залежність Maven:

<dependency>

<groupId>com.google.code.gson</groupId>

<artifactId>gson</artifactId>

<version>2.8.5</version>

</dependency>

Дану залежність необхідно помістити в тег <dependencies>

# Комплексна практична робота 9 Повноцінний додаток «Task Manager»

## Загальні відомості

Після виконання всіх практичних робіт у вашому розпорядженні є бібліотека класів для роботи із календарем задач. Тепер перед вами стоїть задача створити на основі цієї бібліотеки повноцінний додаток, який буде корисний і зручний для користувача.

## Основні вимоги

1. Додаток може мати на вибір студента текстовий (консольний) або графічний (GUI) інтерфейс користувача.
2. Додаток повинен зберігати дані між запусками у файловій системі.
3. Додаток повинен надавати користувачу наступні основні функції:
   1. створювати нові задачі;
   2. змінювати параметри існуючих задач;
   3. видаляти задачі;
   4. переглядати інформацію про існуючі задачі;
   5. переглядати календар запланованих подій на деякий проміжок часу.
4. Додаток повинен сповіщати користувача в момент, коли деяка задача має бути виконана.
5. Додаток повинен бути стійкий до помилок: виводити зрозумілі користувачу повідомлення, не втрачати працездатності, контролювати користувацький ввід.
6. Додаток має використовувати бібліотеку log4j або slf4j для логування.

## Дизайн додатку

Перед тим, як розпочати роботу над кодом необхідно продумати дизайн додатку, результатом якого будуть два основні документи – зовнішній дизайн у вигляді communication діаграми, та внутрішній у вигляді class діаграми.

### Communication Diagram

Ця діаграма повинна показувати які дії може виконувати користувач із системою і до чого це призведе, наприклад:

calendar

1. Початок роботи
2. Список всіх задач
3. Календар на тиждень
4. Введення нової задачі
5. Детальна інформація про задачу
6. Завершення роботи

add/edit ID

save/discard

view ID

activate/deactivaterd

back

quit

back

add

### Class Diagram

Ця діаграма відображає внутрішню архітектуру додатку, системи, підсистеми та модулі, способи взаємодії між ними, наприклад:

Notification subsystem

Model

**Task**  
+ nextTimeAfter(Date): Date  
+ title: String  
+ isActive: boolean

**Tasks**  
+ incoming(List<Task> tasks,   
 Date from, Date to): Set<Task>  
+ calendar(List<Task> tasks,  
 Date from, Date to):   
 Map<Date,Set<Task>>

uses

**NotificationManager**

uses

***<Notification>***+ notify(Set<Task>)

***ScreenNotification***+ notify(Set<Task>)

***MailNotification***+ notify(Set<Task>)

uses

implements

implements

Архітектуру додатку слід проектувати так, щоб окремі підсистеми і модулі були максимально незалежні і взаємодіяли через інтерфейси, бажано використовувати патерни MVC, Observer/EventListener тощо.