МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна система оцінки знань студентів навчально-консультаційного центру Netcracker»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Берест О.Б.**

Студента групи ІН – 61 **Васильєв Є.І.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020 р.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи ІН-61 спеціальності "Інформатика" денної форми навчання Васильєва Євгена Ігоровича.

Тема: "**Інформаційна система оцінки знань студентів навчально-консультаційного центру Netcracker**"

Затверджена наказом по СумДУ

№_____ від _____2020 р.

Зміст пояснювальної записки: 1) вступ, опис існуючої проблеми оцінки знань студентів навчально-консультаційного центру Netcracker; 2) постановка завдання; 3) загальні відомості про Netcracker та середовище розробки Intellij Idea; 4) огляд існуючих систем оцінки знань студентів; 5) реалізація власної системи оцінки знань студентів; 6) висновки.

Дата видачі завдання "_____"_____2020 р.

Керівник випускної роботи _____ Берест О.Б.

Завдання приняв до виконання _____ Васильєв Є.І.

РЕФЕРАТ

Записка: 110 стор., 39 рис., 1 додаток, 9 джерел.

Об'єкт дослідження — система оцінки знань студентів.

Мета роботи — розробка інтелектуальної системи оцінки знань студентів навчально-консультаційного центру Netcracker з предметом вивчення – програмування мовою Java версії 8 або вище.

Методи дослідження — аналіз існуючих систем оцінки знань студентів.

Результати — створено завдання для студентів навчально-консультаційного центру Netcracker та програмне забезпечення, що автоматично перевіряє правильність виконання завдань. Створені завдання охоплюють усі необхідні теми актуальні для мови програмування Java версії 8. Програмне забезпечення являє собою систему автотестів, яка вміє поетапно перевіряти роботу студента. Програмна реалізація була створена у середовищі Intellij Idea, та передбачає інтеграцію з даним середовищем при виконанні завдань студентом.

СИСТЕМА ОЦІНКИ ЗНАНЬ СТУДЕНТІВ, АНАЛІЗ ІСНУЮЧИХ СИСТЕМ ОЦІНКИ ЗНАНЬ, СТВОРЕННЯ ВЛАСНИХ ЗАВДАНЬ, ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ОЦІНКИ ЗНАНЬ, МОДУЛЬНЕ ТЕСТУВАННЯ, JAVA 8, JUNIT 4.

ЗМІСТ

1. ВСТУП

Навчально-консультаційний центр компанії Netcracker (далі НКЦ) планує розпочати новий курс «Програмування мовою Java» з новими завданнями. Планується вивчення усіх основних тем, що відносяться до Java SE. Для оцінки знань необхідно мати систему, за якою студент і викладач може об’єктивно оцінити виконану роботу.

Зараз НКЦ має у своєму розпорядженні систему автотестів, що автоматизує перевірку правильність виконання завдань студентами. Викладач на основі результатів проходження таких тестів може оцінювати правильність роботи алгоритму, що створив студент. Отже, для повної оцінки йому залишається лише поглянути на стиль коду та перевірити теоретичні знання студента.

На жаль, існуюча система автотестів не відповідає актуальним стандартам вивчення мови програмування Java. Вона була розрахована на вже неактуальну версію Java, а завдання для студентів не включають ті нововведення мови, які зараз уже є звичною справою для роботи на професійному рівні. Крім того, виконання автотестів відбуваються у середовищі Notpad++, що майже унеможливлює налагодження програми у разі неочікуваного результату виконання.

Курс «Програмування мовою Java» спрямований на вивчення актуальних технологій. Після завершення курсу студент повинен отримати необхідні знання Java Standard та Enterprise edition. Тому необхідно, щоб завдання, що виконує студент, відповідали актуальній версії Java. На стан 2020 року оптимальна версія – 8 або вища.

Дана робота присвячена створенню інформаційної системи оцінки знань студентів з курсу «Програмування мовою Java», що проводить НКЦ на базі СумДУ.

У даній роботі буде розглянуто існуючи системи оцінки знань, їх переваги і недоліки. З урахуванням проаналізованих даних будуть створені власні завдання, а також буде запропонований принцип їх перевірки і оцінювання.

2. ПОСТАНОВКА ЗАДАЧІ

Необхідно розглянути існуючі системи оцінки знань студентів з курсів мов програмування. Потрібно здійснити їх аналіз та визначити ключові моменти, за якими будуть створюватись нові завдання для курсу НКЦ. Також потрібно визначити, яким чином буде реалізований аналіз якості виконуваних завдань студентом.

Необхідно створити програмне рішення, що буде перевіряти правильність виконуваних завдань. Програма повинна ураховувати етап, на якому знаходиться студент, та створювати необхідний список задач, що будуть перевірятись (студент не повинен бачити повідомлення про непройдені тести, якщо він ще не дійшов до даної теми тестів).

Необхідно дати змогу студентові виконувати поставлені задачі у середовищі Intellij Idea. Сам проект повинен бути зібраний на базі фреймворку Maven.

Головною вимогою є відповідність завдань актуальній версії Java – 8 або вище.

3. ЗАГАЛЬНІ ВІДОМОСТІ

Netcracker – дочірня компанія корпорації NEC. Галузі в яких працює Netcracker – телекомунікації та інформаційні технології. Компанія спеціалізується у створенню, впровадженні і супроводі систем підтримки операцій, систем підтримки бізнесу, а також рішень віртуалізації для операторів зв’язку, великих підприємств і державних установ.[1]

Компанія має власний НКЦ, що на базі СумДУ проводить безкоштовні курси QA, TA і Java Programming. Найбільш успішні випускники НКЦ Netcracker будуть запрошені працювати в Netcracker з гнучким графіком для студентів.

Виконуючи практичні завдання студент буде працювати з відповідним середовищем для розробки програмних рішень Intellij Idea. Дане середовище створено компанією JetBrains у 2001-му році. Хоча середовище отримало широку популярність серед Java розробників, воно також підтримує роботу з усіма найбільш популярними мовами програмування, серед яких Python, JavaScript, PHP, HTML та ін. Головною перевагою Intellij Idea є надзвичайно широкі можливості у рефакторингу коду, а також автодоповнення. Ця сукупність значно скорочує час розробки програмних рішень шляхом автоматизації та генерації вихідного коду. [2]

4. ОГЛЯД ІСНУЮЧИХ СИСТЕМ ОЦІНКИ ЗНАНЬ

Зараз існує досить велика кількість курсів з програмування. Деякі з них проходяться у режимі онлайн, деякі проводять різні компанії у лекційних приміщеннях. Курси мають різні системи оцінки знань студентів. Але є декілька схожих рис, які притаманні великій кількості системам.

Так, наприклад, у більшості курсах, що проходять онлайн, існує система, за якою студент після завершення вивчення матеріалу (як всього, так і окремої теми), повинен пройти деяке тестування, або йому дається тестове завдання, яке він повинен виконати. В останньому випадку студент відправляє завдання системі, і воно автоматично (або вручну відповідною людиною) перевіряється. Також можливе обмеження у часі на виконання. Перевагою даних курсів є їх велика різноманітність, що пропонує великий вибір для студентів. Причому багато з них є безкоштовними.

Для тих курсів, що проводяться у якості очної зустрічі, також є характерним складання тестів або виконання тестових завдань. Але тут уже більш переважає практика обмеження у часі. Очевидною перевагою таких курсів є можливість поставити запитання з високою ймовірністю швидкого отримання відповіді, та обговорювати деталі теми з відповідною кваліфікованою людиною. Більшість подібних курсів є платними.

Кожен з курсів різного типу мають свої недоліки і переваги. Також варто мати на увазі те, що якщо існує деякий платний курс з програмування, то він не завжди буде кращим за відповідний безкоштовний. Далі буде розглянуто найбільш популярні курси.

Stepik

Stepik — освітня платформа з відкритими безкоштовними онлайн-курсами та уроками у режимі онлайн.

Даний сервіс дає змогу будь-якому зареєстрованому користувачеві створити інтерактивні уроки або онлайн-курси, використовуючи відео, тексти й різні задачі з автоматичною перевіркою та миттєвим зворотнім зв'язком. У процесі навчання можна вести обговорення між учасниками курсу та ставити питання викладачу на форумі. Основні дисципліни, які охоплені курсами — це програмування, математика, біоінформатика і біологія, економіка; основна мова курсів — російська, а також є курси англійською мовою. Станом на червень 2018 року платформа містила зареєстрованих 1 мільйон користувачів, з яких 77 % з Росії, 9 % — України, 3 % — Республіки Білорусь, 2 % — США, 1 % — Казахстану. Цільова аудиторія — школярі (в основному курси з підготовки до ЗНО), студенти, початківці спеціалісти.[3]

Тут неможливо виділити окремий курс серед інших, бо тут знаходяться курси як компаній, так і звичайних викладачів. Більшість курсів на платформі Stepik являють собою видачею освітнього матеріалу з прикладами. Після вивченої теми студенти проходять тестування та виконують практичні завдання без обмежень у часі. Завдання перевіряються автоматично системою. Після завершення курсу студент отримує сертифікат.

Переваги:

- Безкоштовний;
- у більшості випадків - хороша підтримка та зворотний зв’язок;
- підтримка великих ІТ-компаній;
- комбінація тестів і практичних завдань;

- великий вибір курсів для студентів різних категорій (для початківців і тих, що вже мають досвід у деякій області).

Недоліки:

- не всі курси можуть відповідати очікуванням.

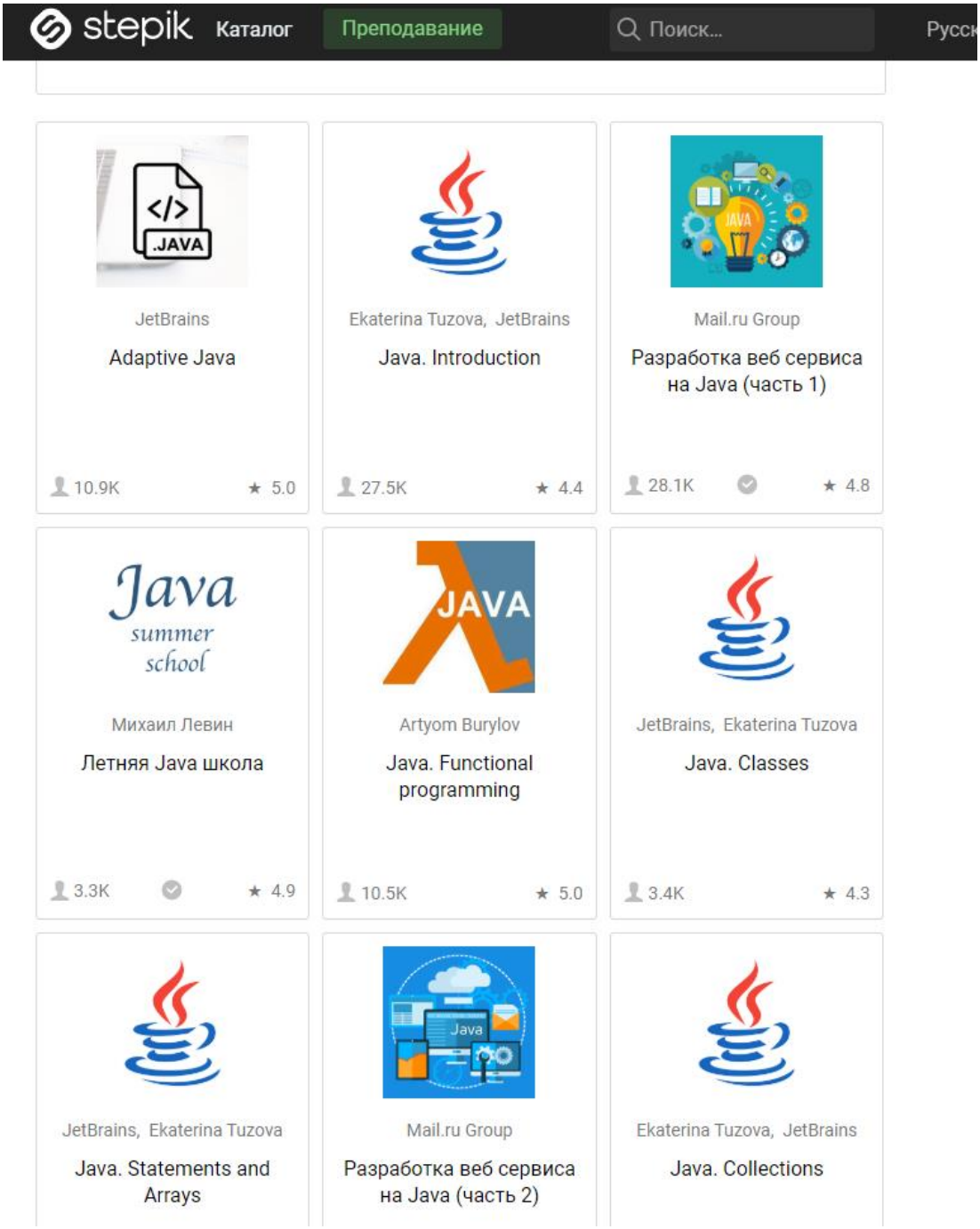

Рисунок 1 – вибір курсів на сайті

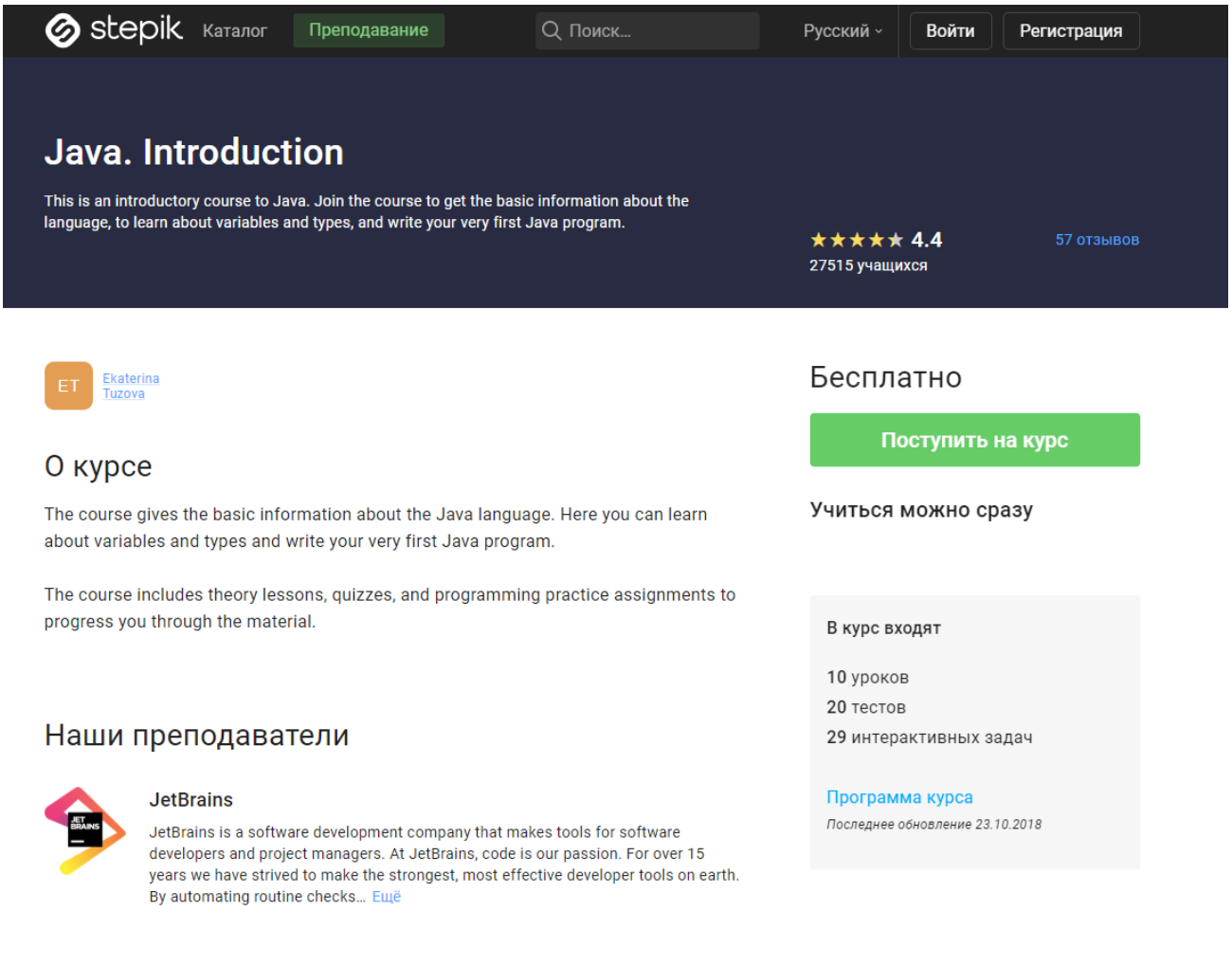

Рисунок 2 – приклад курсу

JavaRush

JavaRush - це онлайн сервіс для навчання програмування мовою Java. Курс навчання створений у вигляді гри за мотивами всесвіту Футурами. Мета гри - прокачати персонажа (робота Аміго) з 1 до 80-го рівня. У грі користувач виконує завдання і заробляє чорну матерію, яка витрачається на відкриття нових рівнів.

Користувач бере завдання, виконує його та отримує нагороду. Завдання будуть найрізноманітніші: читання коду, розв’язання задач, відеоуроки, перегляд Футурами, виправлення помилок в коді, додавання нових фіч і багато іншого.

Даний курс містить велику кількість матеріалу і практичних завдань. Він охоплює усі теми з Java SE, а також пропонує продовжити навчання та перейти до Java EE. Після кожної теми студентові пропонується виконати невелике завдання на вебсторінці, яке одразу буде перевірено. Даний курс позиціюється як такий, що має великий акцент на практичні завдання.

Переваги:

- навчання у формі гри;
- доступна подача матеріалу;
- велика кількість практичних завдань;
- хороша підтримка.

Недоліки:

- курс платний.

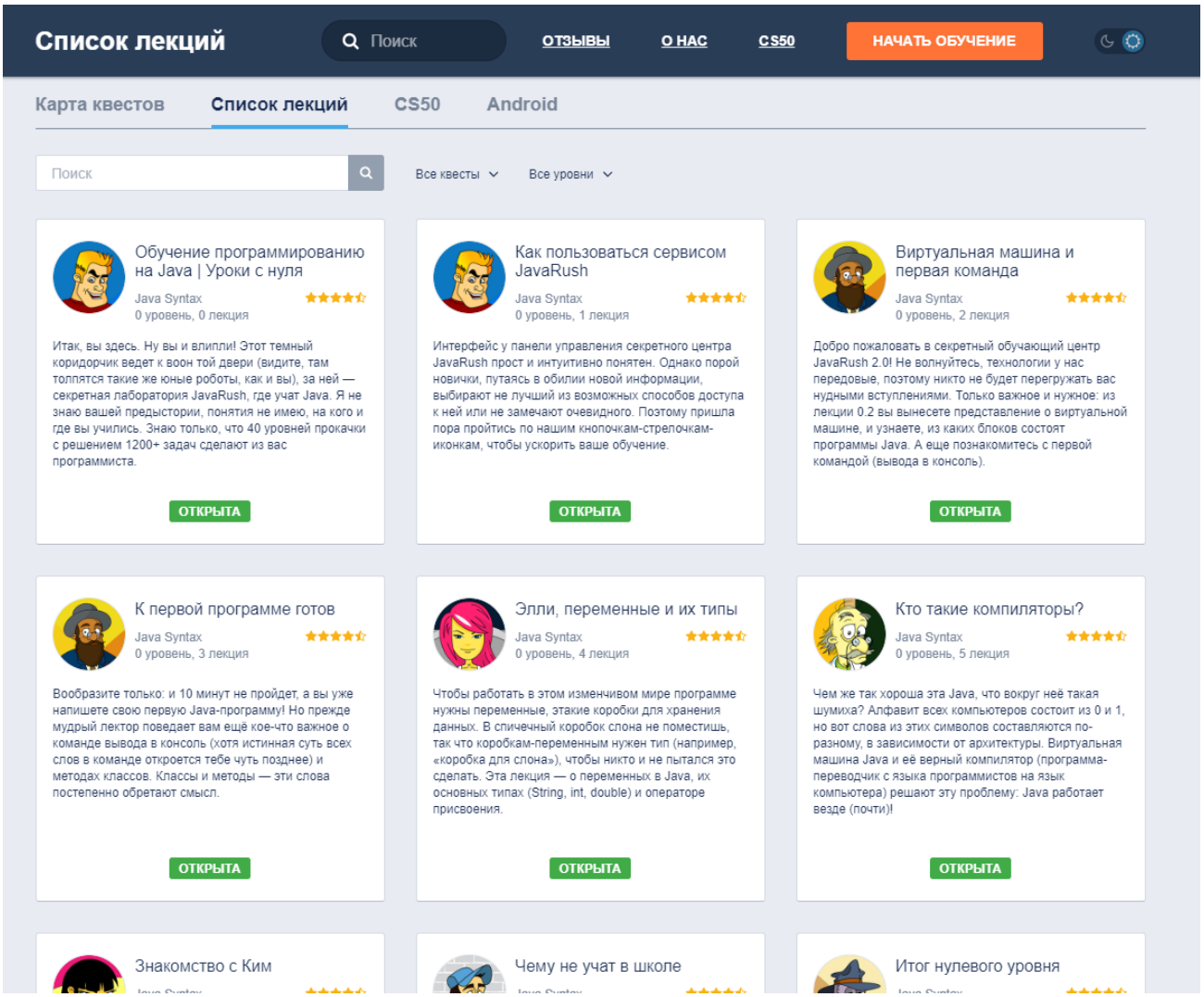

Рисунок 3 – вибір лекцій

SoloLearn

Подібно до Stepik, SoloLearn являє собою платформу для створення і проходження курсів. Вона зберігає колекцію безкоштовних матеріалів по вивченню програмування для новачків і професіоналів. Тисячі аспектів програмування допоможуть освоїти ази, відточити навички або просто бути в курсі останніх тенденцій. Кожен день спільнота SoloLearn створює нові навчальні матеріали, щоб ви могли швидко і ефективно відточувати свої навички. Мільйони програмістів допоможуть освоїти безліч тем і завдань з будь-якої точки світу в будь-який час.

Проект моє власний мобільний додаток, тому студенти мають змогу з будь-якого пристрою у зручному форматі переглядати матеріал. На курсах паралельно з видачею матеріалу студенти проходять тест по щойно опрацьованій темі. Після закінчення курсу студент отримує сертифікат.

Переваги:

- велика кількість матеріалу;
- активна спільнота з великою кількістю дискусій;
- мобільний додаток;
- повністю безкоштовне.

Недоліки

- курси не передбачають практичні завдання;
- матеріали курсу мають оглядовий характер, багато тем пропущено.

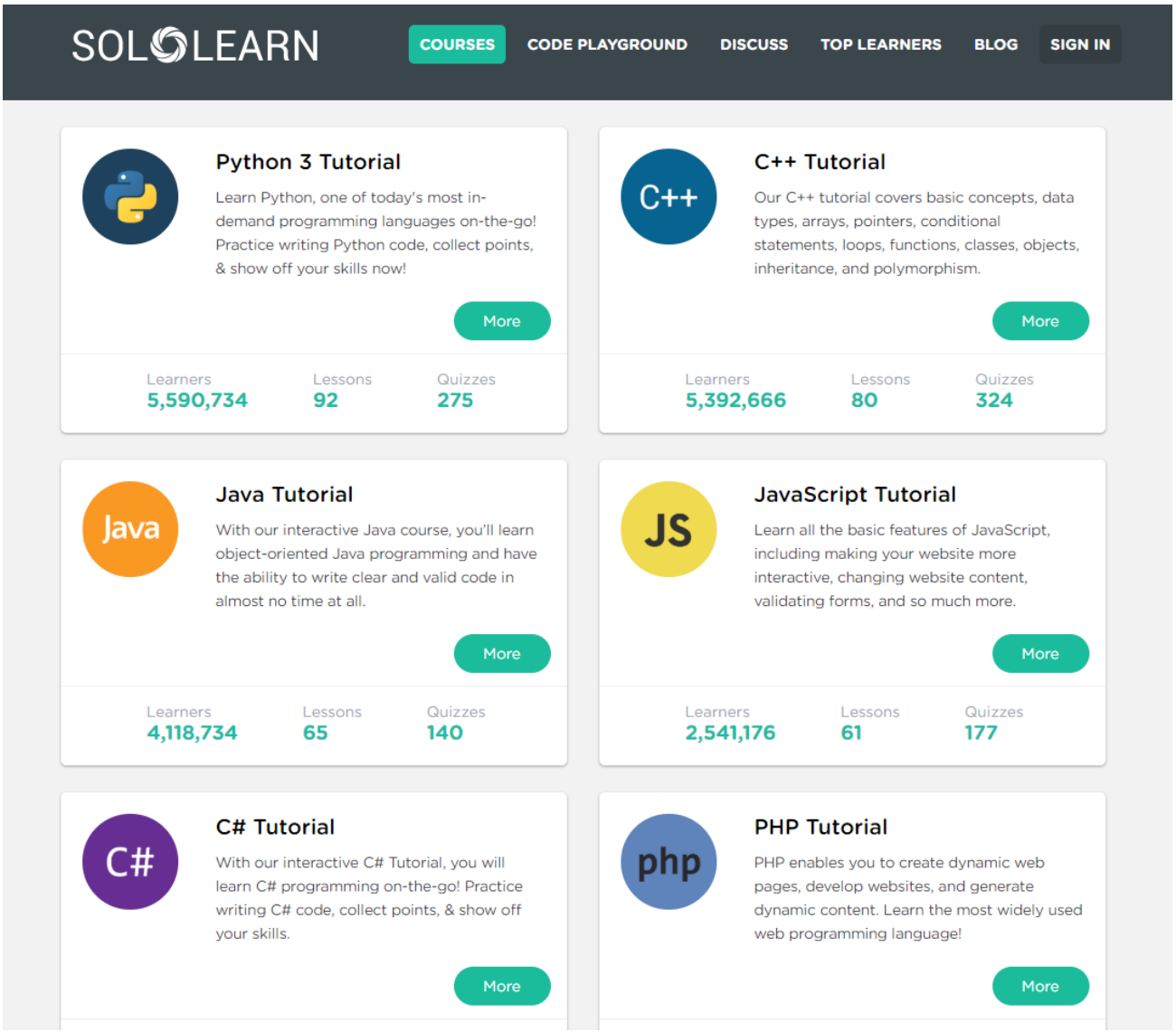

Рисунок 4 – вибір курсів

5. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОЦІНКИ ЗНАНЬ

5.1. Створення оновлених завдань

Курс Програмування мовою Java триває два семестри. Увесь перший семестр присвячений Java SE. Під час нього студенти поступово виконують лабораторну роботу №1. Курс складається з лекційних і практичних завдань. Завдання роботи було створено так, щоб поступово перейти від основ ООП до потоків вводу-виведення.

Для оцінки правильності виконання завдань будуть створені автотести.

Всього буде 8 практичних завдань, розподілені за темами. На Рисунок 5 зображено назви праткичних завдань.

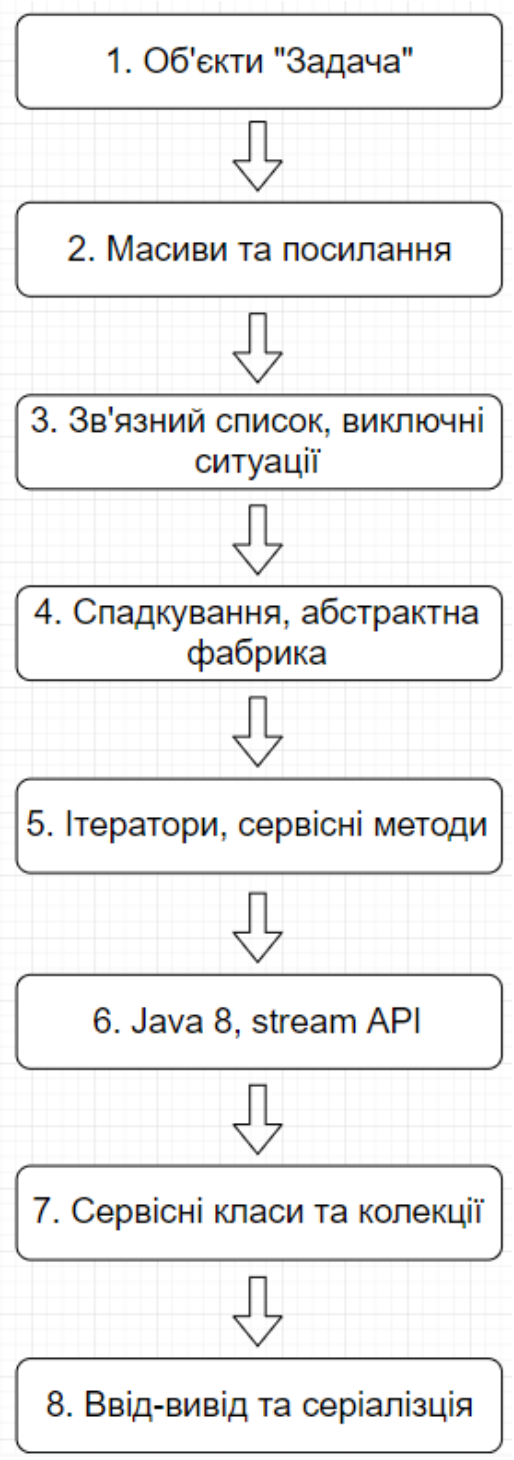

Рисунок 5 – теми практичних завдань

Розглянемо більш детально кожне із завдань.

5.1.1. Практична робота 1

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:

  a) int getTime(), у разі, якщо задача повторюється метод має повертати час початку повторення;
  b) 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:

  a) int getStartTime(), у разі, якщо задача не повторюється метод має повертати час виконання задачі;

b) int getEndTime(), у разі, якщо задача не повторюється метод має повертати час виконання задачі;
c) int getRepeatInterval(), у разі, якщо задача не повторюється метод має повертати 0;
d) 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.

ЧАСТИНА 2

Необхідно додати у клас 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.

5.1.2. 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 зі 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 цей 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.

5.1.3. Практична робота 3

ЗАВДАННЯ 1. ЗВ’ЯЗНИЙ СПИСОК

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 та з такими самими методами, що і ArrayTaskList (метод incoming змінить тип об’єкту, що повертається). Об’єкти цього класу мають поводити себе так само, як і об’єкти класу ArrayTaskList.

ЗАВДАННЯ 2. 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. Рішення щодо того, що саме робити у випадках порушення обмежень і якщо породжувати виключення, то якого типу повинні бути аргументовані.

5.1.4. Практична робота 4

ЗАВДАННЯ 1. СПАДКУВАННЯ

1. Оскільки обидва класи ArrayTaskList і LinkedTaskList реалізовують один і той самий тип даних необхідно створити абстрактний клас TaskList, де описати методи, характерні для списку задач як абстрактні та успадкувати обидва класи списків від цього класу.

2. Оскільки реалізація методу incoming не залежить від способу зберігання задач, цього можна реалізувати в абстрактному класі TaskList (при цьому тип об’єкту, що ним повертається також зміниться на TaskList), таким чином зменшивши дублювання коду.

ЗАВДАННЯ 2. АБСТРАКТНА ФАБРИКА

На даному етапі ми маємо два класи ArrayTaskList і LinkedTaskList, і спільний для них батьківський клас TaskList. Було б досить зручно мати окремий клас, який би мав метод, що створює об’єкт ArrayTaskList і LinkedTaskList в залежності від параметра, що в нього передається. Саме в цьому і є ідея абстрактної фабрики.

1. Створіть клас ListTypes з полем types типу enum. Types повинно містити значення ARRAY та LINKED.

2. Створіть клас TaskListFactory зі статичним методом TaskList createTaskList(ListTypes.types type). Даний метод, відповідно до параметра type, повинен повертати об'єкт класу ArrayTaskList або LinkedTaskList.

5.1.5. Практична робота 5

ІТЕРАТОРИ, СЕРВІСНІ МЕТОДИ ЗАВДАННЯ

ЗАВДАННЯ 1. ІТЕРАТОРИ

Для абстрактного опису об’єктів, що складаються із послідовності об’єктів іншого типу у стандартній бібліотеці існує інтерфейс java.util.Iterable, 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 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 в одному 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 та списки задач повинні дозволяти клонувати себе. Реалізація не повинна залучати виклик конструктора класу.

У цьому завданні необхідно імплементувати інтерфейс Iterable для списків та перевизначити базові методи із класу Object.

5.1.6. Практична робота 6

Java 8

Починаючи з версії Java 8 стали доступними лямбда-вирази та stream API. Лямбда-вираз – це анонімна функція, вона не має свого імені, може отримати доступ до змінних своєї області видимості. stream API є потужним інструментом для операцій з елементами колекції. Даний інструмент дозволяє ітеруватись по колекції елементу та виконувати такі операції, як фільтрація, пошук максимуму/мінімуму/суми для елементів, збір елементів у іншу колекцію або в рядок з символами-роздільниками.

1. Додайте до класу TaskList метод Stream<Task> getStream(). Даний метод повинен бути перевизначений у дочірніх класах.

2. Перенесіть реалізацію методу incoming(int from, int to) із класів ArrayTaskList і LinkedTaskList в TaskList. Опишіть логіку методу incoming(int from, int to) в TaskList з використанням stream API, що з'явився в Java 8. Забороніть методу incoming(int from, int to) бути перевизначеним у дочірніх класах, він повинен мати реалізацію лише у класі AbstractTaskList.

5.1.7. 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 починаючи з версії Java 8 у стандартній бібліотеці існує клас LocalDateTime для представлення дати та часу. Необхідно замінити використання цілих чисел на LocalDateTime для роботи із моментами часу.

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

ЗАВДАННЯ 2. ВИДІЛЕННЯ ЛОГІКИ РОБОТИ З КАЛЕНДАРЕМ

На цей 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 списків задач – сигнатура методу тепер буде Iterable<Task> incoming(Iterable tasks, LocalDateTime start, LocalDateTime end).

Окрім того у класі Tasks необхідно реалізувати ще один статичний метод SortedMap<LocalDateTime, Set<Task>> calendar(Iterable 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.

5.1.8. Практична робота 8

ВВІД-ВИВІД ТА СЕРІАЛІЗАЦІЯ

Для реалізації повноцінної програми по роботі з задачами необхідно дозволити зберігати задачі та списки задач на диску та передавати мережею. 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 зі статичними методами:

- void write(TaskList tasks, OutputStream out) – записує задачі зі 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 зі списку у файл.

- void readBinary(TaskList tasks, File file) – зчитує задачі із файлу у список 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 текстовому форматі.

- void read(TaskList tasks, Reader in) – зчитує задачі із потоку у список.

- void writeText(TaskList tasks, File file) – записує задачі у файл у текстовому форматі

- void readText(TaskList tasks, File file) – зчитує задачі із файлу у текстовому вигляді. Текстовий формат у всіх випадках повинен бути JSON – один з найпростіших та читабельних форматів зберігання даних.

Для цього рекомендується використовувати бібліотеку GSON, яку можна включити в проект, використовуючи залежність Maven:

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.5</version>
</dependency>
```

Скопіюйте дану залежність у файл pom.xml вашого проекту у блок <dependecies>

Дані завдання охоплюють новий функціонал Java 8, бібліотеку для запису даних у форматі JSON та шаблон проектування абстрактна фабрика.

5.2. Створення автотестів

Тепер розглянемо схему, за якою будемо створювати оновлені автотести для нашого завдання. Перш за все, проект буде збиратись за допомогою фреймворку Maven – найбільш поширений інструмент керування залежностями та збирання Java проектів.[4][5]

У якості автотестів будуть виступати юніт-тести. Дані тести дозволяють тестувати кожний модуль коду програми (кожний метод) окремо, незалежно один від одного. Якщо один з методів не пройшов тестування, тестування для інших методів не зупиниться. Даний спосіб найбільше підходить для перевірки виконання наших завдань, які передбачають поступове додавання до проекту нових класів та методів – нам потрібно протестувати лише ті методи, які є у проекті.[6]

Для реалізації юніт-тестів будемо використовувати бібліотеку JUnit 4. Дана бібліотека одна є однією з найпростіших, містить мінімально необхідний функціонал для проведення юніт-тестів.[7]

Додамо бібліотеку в проект за допомогою залежності maven:

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

Головною проблемою на даному етапі є те, що проект не буде збиратись, якщо не будуть існувати класи, які необхідно тестувати. Іншими словами, без тестових класів не можуть існувати юніт-тести.

Бібліотека Junit не передбачає можливість перевіряти наявність тестових класів на етапі виконання програми. Тож було вирішено зробити свою реалізацію перевірки наявності класів за допомогою рефлексії – стандартний інструмент Java, що дозволяє маніпулювати класами без необхідності створення їх об’єктів.

Структура проекту, в якому студенти будуть виконувати завдання має наступний вигляд, зображена на Рисунок 6.

Рисунок 6 – Структура проекту

У пакеті ua.sumdu.j2se.studentName.tasks студенти будуть створювати свої власні класи з логікою, що відповідає завданням представленим вище.

Для автотестів найбільш важливим є пакет com.netcracker.eductr.tasks.tests. У ньому знаходяться всі класи, що будуть залучені у виконанні автотестів.

Перш за все перед початком виконання автотестів необхідно перевірити наявність необхідних класів та їх методів, що будуть протестовані.

Бібліотека JUnit містить спеціальні анотації, що дозволяють керувати порядком виклику тих чи інших методів. Для нашої реалізації знадобиться анотація @BeforeClass. Дана анотація говорить, що цей метод буде викликаний першим серед усіх інших методів. Ми використаємо цю поведінку, щоб перевіряти наявність класів.[8]

Лістинг усіх класів можна знайти у Додатку А. Далі буде розглянуто лише структуру класів.

5.2.1. Модель

Перший пакет з класами, який ми опишемо, буде com.netcracker.eductr.tasks.tests.model.*. У даному пакеті будуть зберігатись основні класи, з якими будуть працювати автотести. Дані класи будуть виступати у ролі посередників між автотестами і класами, що створе студент. Вони за допомогою рефлексії будуть викликати необхідні нам методи класів. Ієрархія класів моделі зображена на Рисунок 7.

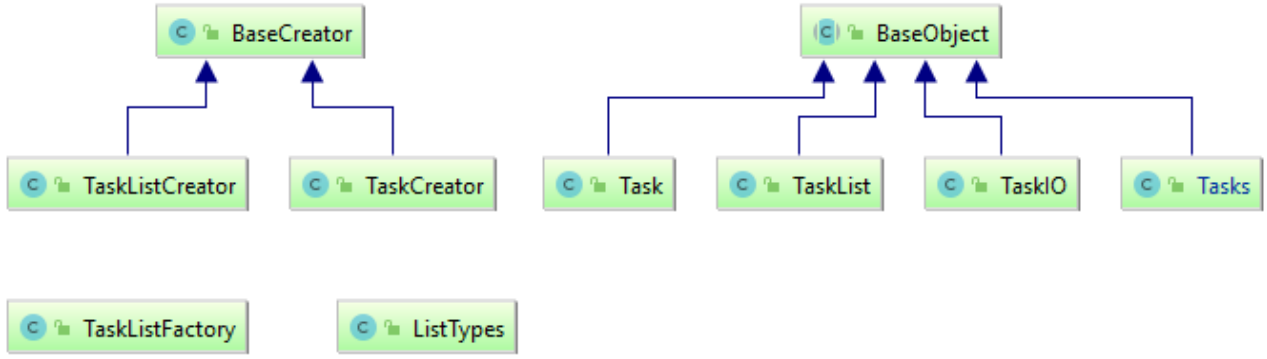

Рисунок 7 – Ієрархія класів моделі

Спочатку створимо базовий клас BaseObject. Цей клас є батьківським для усіх класів моделі, та містить спільний для всіх метод, що повертає об’єкт відповідного класу. (Рисунок 8)

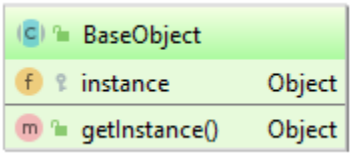

Рисунок 8 – Структура класу BaseObject

На Рисунок 9 зображено структуру класу Task з пакету com.netcracker.eductr.tasks.tests.model. Цей клас буде мати методи що керує однойменним класом, що створе студент (створення об’єктів, виклик методів, і т.д.). Таким чином якщо ClassFinder#checkClassExistence поверне true (Тобто студент у своєму пакеті створив відповідний клас Task), то ми можемо використовувати допоміжний Task для виклику відповідних методів і виконання тестів.

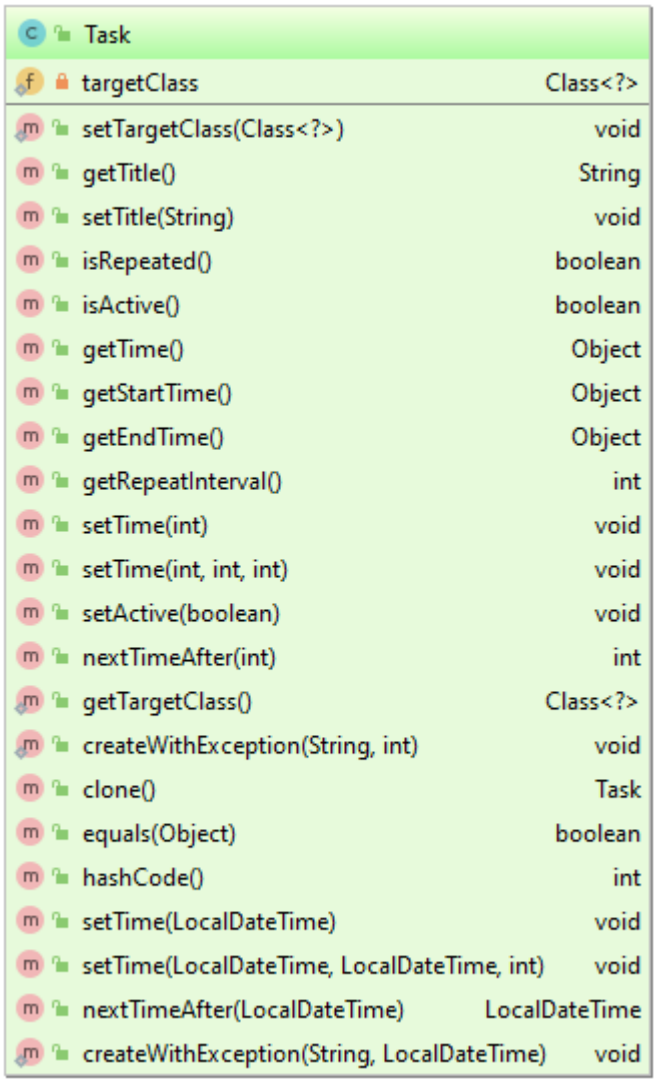

Рисунок 9 – Структура класу Task

Даний клас буде викликати конструктор і методи справжнього класу Task, який будуть створювати студенти курсу. Клас успадкований від BaseObject.

Наступний клас буде TaskList. Він буде звертатись до класів, що реалізують списки. (Рисунок 10)

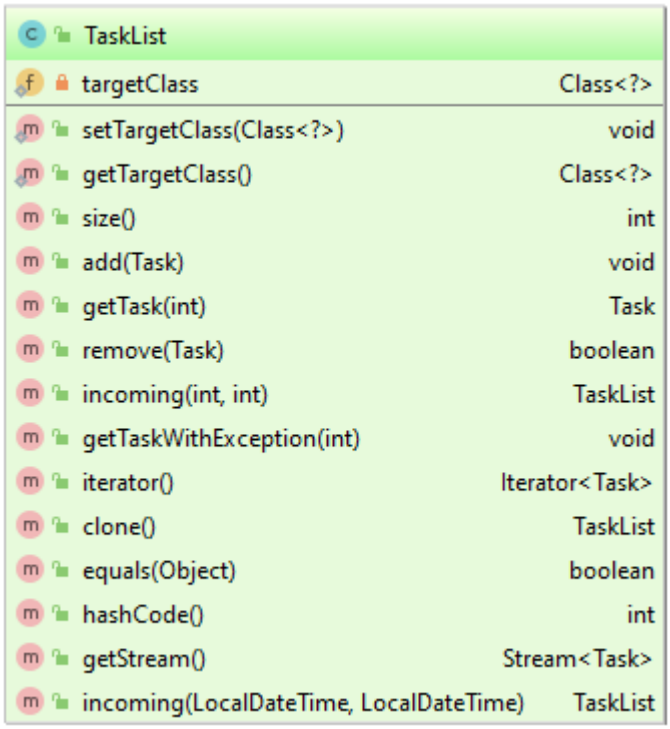

Рисунок 10 – Структура класу TaskList

Створимо клас TaskIO. Він буде відповідальним за однойменний клас для файлового читання/запису. (Рисунок 11)

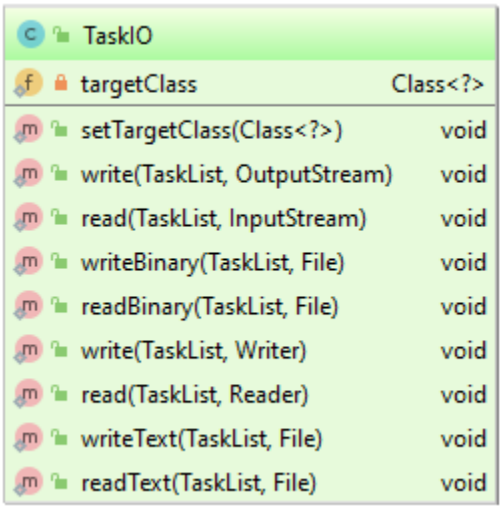

Рисунок 11 – Структура класу TaskIO

На Рисунок 12 зображено структуру класу Tasks, що виконує роль календаря з задачами.

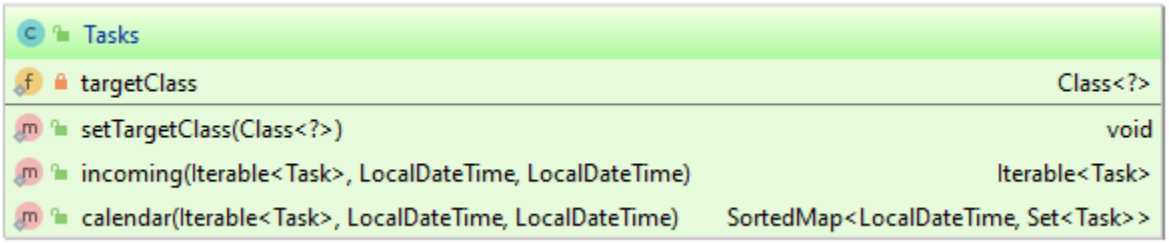

Рисунок 12 – Структура класу Tasks

Для пришвидшення написання автотестів створимо допоміжні класи, що будуть створювати для нас об’єкти класів задач і списків задач. Це також допоможе нам позбутися повторюваності коду.

Першим таким класом буде BaseCreator, структуру якого зображено на Рисунок 13. Він буде основою для інших класів *Creator, та містить у собі поле IS_OLD. Завдяки цьому полу можна буде дізнатись на якому етапі зараз знаходиться студент: використовує він для позначення часу цілі числа, або ж відповідні класи з пакета java.time.*.

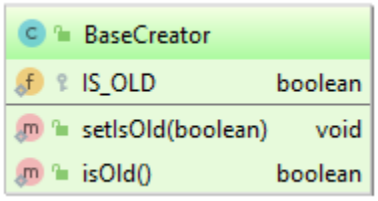

Рисунок 13 - Структура класу BaseCreator

Далі створимо клас TaskCreator. Він містить множину перевантажених методів з різними типами параметрами. Далі в коді ми використаємо ці методи для створення об’єктів задач з наперед вставленими атрибутами (назвою, час почату, тощо). (Рисунок 14)

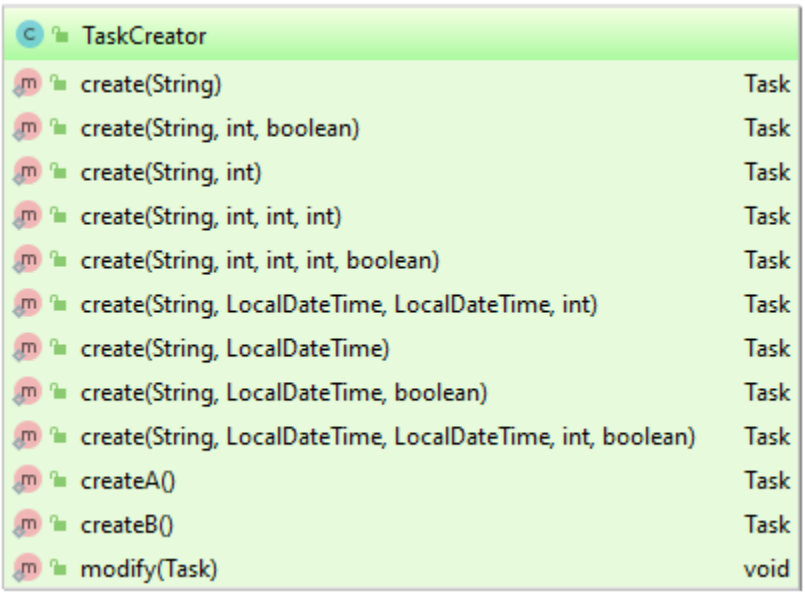

Рисунок 14 - Структура класу TaskCreator

Тепер створимо клас TaskListCreator. Він буде створювати та модифікувати вже списки задач. Структу класу зображено на Рисунок 15.

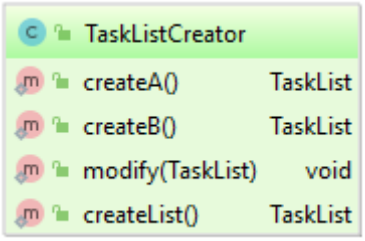

Рисунок 15 - Структура класу TaskListCreator

Наступним буде клас TaskListFactory. Він відповідає однойменному класу з одного з завдань курсу. (Рисунок 16)

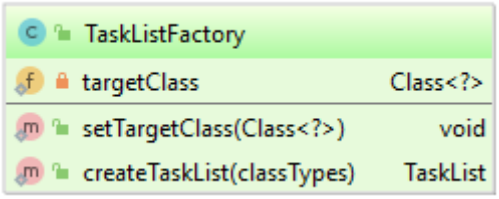

Рисунок 16 - Структура класу TaskListFactory

Залишилось створити клас LisTypes. Цей клас буде використовуватись у TaskListFactory, щоб створити список задач з необхідною реалізацією: масив або зв’язний. (Рисунок 17)

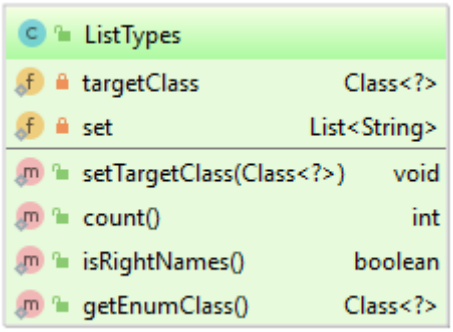

Рисунок 17 - Структура класу ListTypes

5.2.2. Утилітарні класи

У цьому розділі розглянемо утилітарні класи. Дані класи будуть виконувати необхідні перевірки існування класів, їх методів, а також взаємодію одних класів з іншими.

Спочатку розглянемо класи з пакета com.netcracker.eductr.tasks.tests.utils.*.

Першим буде клас ClassFinder, структуру якого зображено на Рисунок 18. Він містить методи з необхідною логікою для перевірки наявності класів і їх методів у пакеті студента. Значення, які повертають методи з цього класу грають вирішальну роль у рішенні запускати даний тест чи ні. Відсутність реалізації студента фіксується цим класом і, як результат, перебіг тестів не зупиняється (студент не зустрічає у консолі повідомлення про неочікувані виключення).

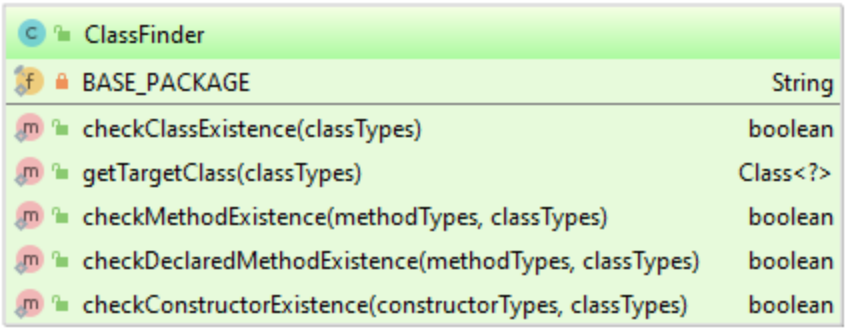

Рисунок 18 - Структура класу ClassFinder

Допоміжним для класу ClassFinder є клас Types. Він містить у собі переліки (enumeration), які використовуються при зверненні до тих чи інших методів студента через рефлексію. (Рисунок 19)

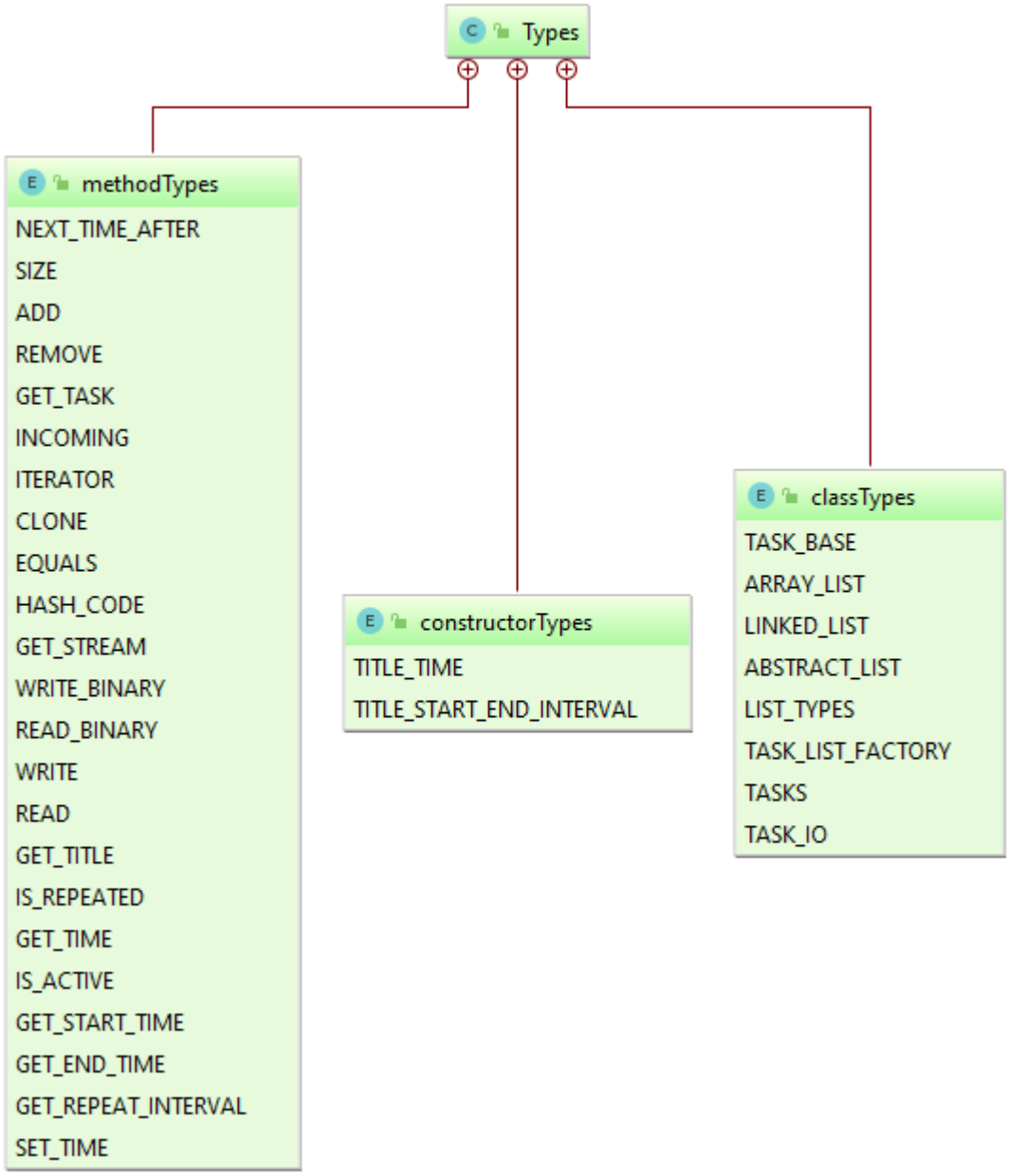

Рисунок 19 - Структура класу Types

Наступним є клас TaskUtil, який використовується для представлення об’єктів задач студента у текстовому вигляді. Структуру класу можна знайти на Рисунок 20. Функціонал класу застосовується при виведенні повідомлень непройдених тестів у зручному для студента вигляді.

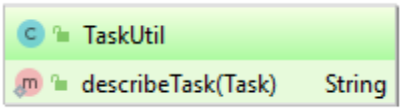

Рисунок 20 - Структура класу TaskUtil

Аналогічним до TaskUtil є клас TaskListUtil, але тепер він працює для списків задач. На Рисунок 21 зображено його структуру.

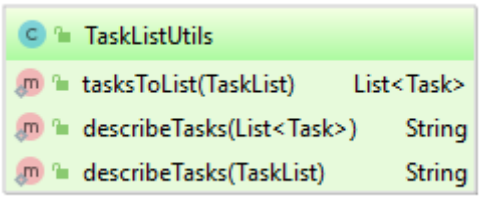

Рисунок 21 - Структура класу TaskListUtils

На Рисунок 22 зображено клас EqualsUtils, що безпосередньо використовується у всіх тестах, де необхідна перевірка рівності об’єктів і списків задач.

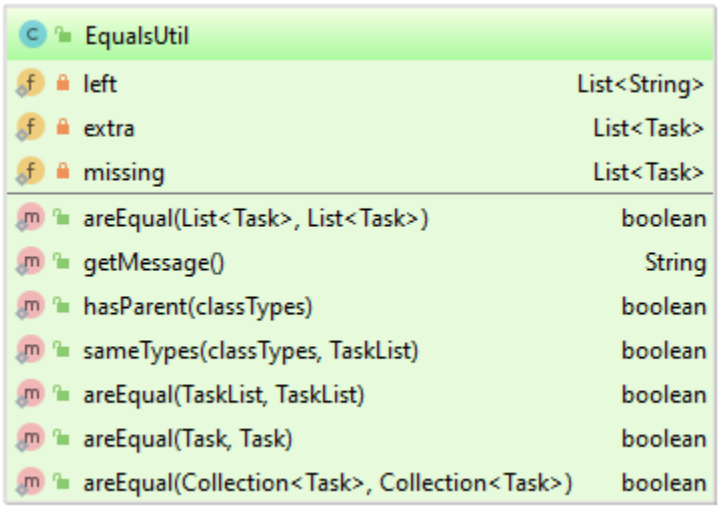

Рисунок 22 - Структура класу EqualsUtil

Ще одним допоміжним класом є DatesTimes. У ньому зберігаються константі значення часу, що використовуються у тестах. (Рисунок 23)

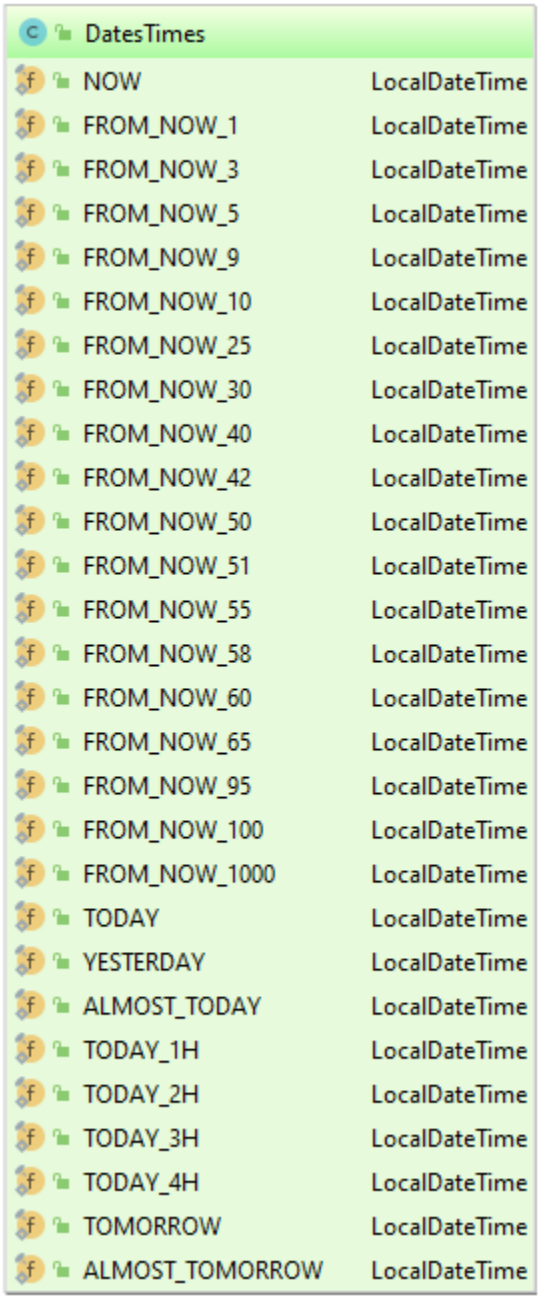

Рисунок 23 - Структура класу DatesTimes

У пакеті com.netcracker.eductr.tasks.tests.asserts створимо клас AssertsExt, зі структурою представленою на Рисунок 24. Даний клас розширяє можливості JUnit 4 та створює нові методи, що задовольняють наші потреби, а саме – перевірка рівності списків задач.

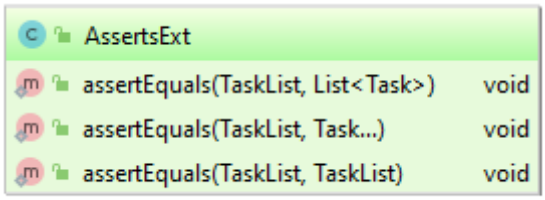

Рисунок 24 - Структура класу AssertsExt

5.2.3. Тестові класи

Далі розглянемо класи, що виконую тестування класів студента. Класи поділені на теми, що відповідають оновленим завданням. До однієї теми можуть відноситись декілька класів. Це означає, що ці класи містять різний характер тестів. На рисунках 25-32 зображено структури тестових класів для відповідних лабораторних робіт.

Рисунок 25 - Структура класів для перевірки першої лабораторної роботи

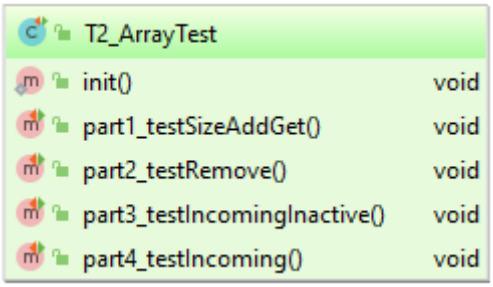

Рисунок 26 - Структура класу для перевірки другої лабораторної роботи

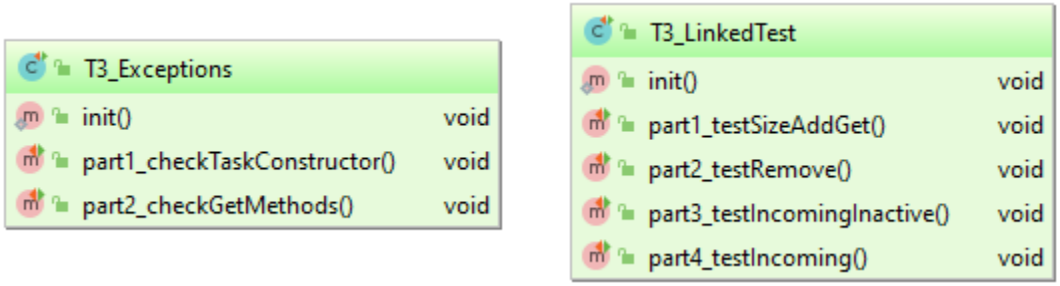

Рисунок 27 - Структура класів для перевірки третьої лабораторної роботи

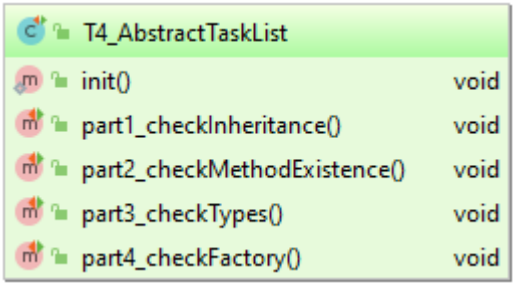

Рисунок 28 - Структура класу для перевірки четвертої лабораторної роботи

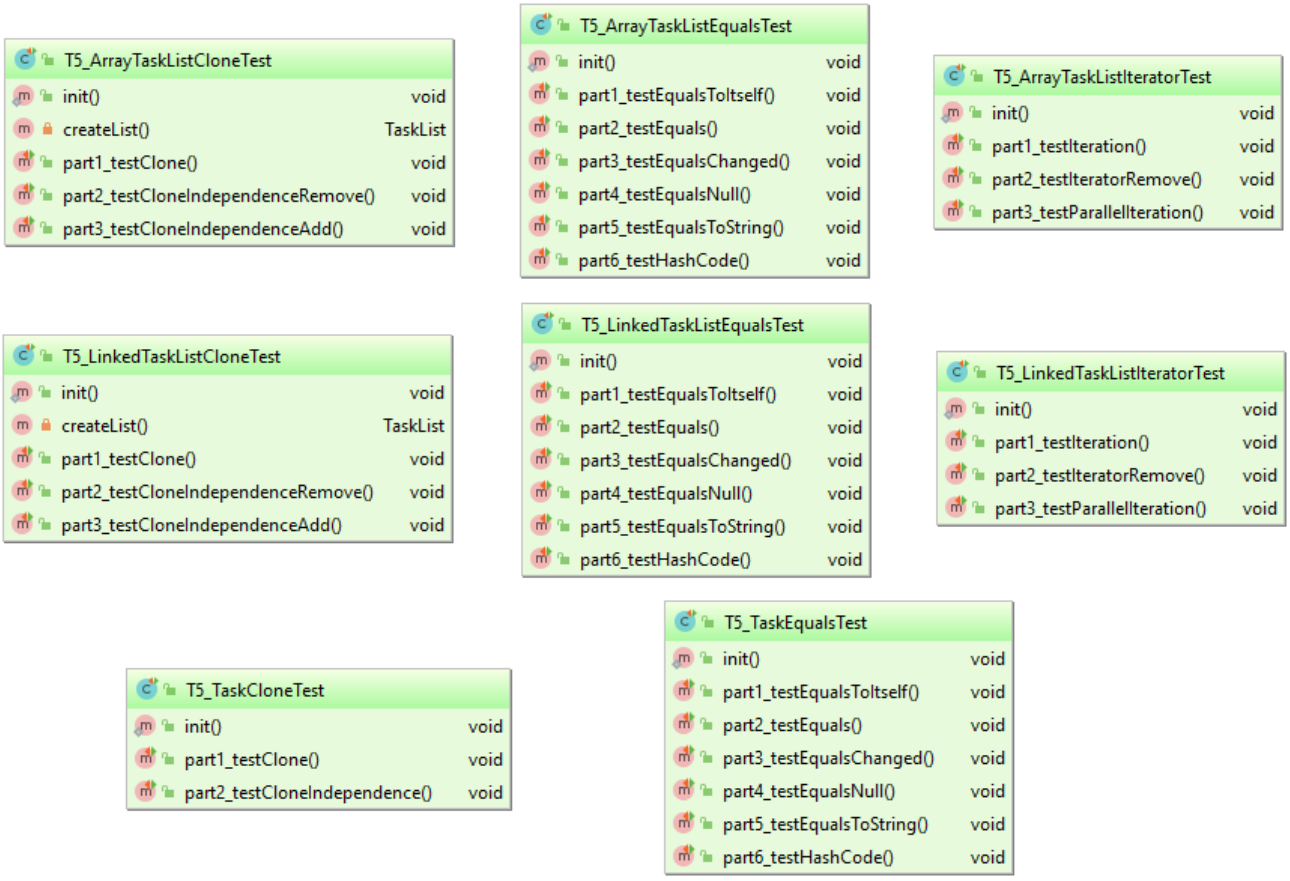

Рисунок 29 - Структура класів для перевірки п’ятої лабораторної роботи

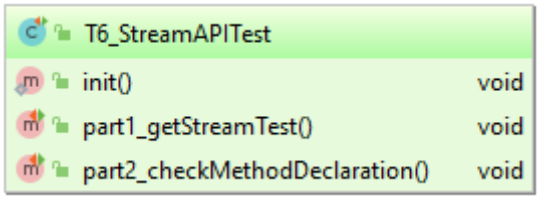

Рисунок 30 - Структура класу для перевірки шостої лабораторної роботи

Рисунок 31 - Структура класів для перевірки сьомої лабораторної роботи

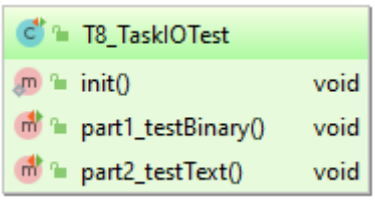

Рисунок 32 - Структура класу для перевірки восьмої лабораторної роботи

У всіх класах вище є спільний метод init маркований анотацією @BeforeClass. Отже, цей метод буде викликаний першим і він перевірить наявність відповідного класу, перш ніж розпочати тестування.

Таким чином, якщо студент ще не створив клас Task, то даний тест буде проігнорований, але проект біде зібраний і студент не зіткнеться з проблемами компіляції.

За кожний пройдений тест студент буде отримувати рейтинговий бал. Бали студентів будуть збиратись в єдину таблицю, де можна буде прослідковувати успішність кожного з них.

5.3. Інструкція користування системою оцінки знань

У цьому розділі опишемо, як студент буде користуватися автотестами, які для цього необхідно виконати налаштування та як вирішувати завдання.

Студент отримає в електронному вигляді список завдань аналогічним у пункті 5.1 та архів з шаблоном проекту.

Архів, вміст якого можна побачити на Рисунок 33, складається зі структури папок, що містять вихідний коду автотестів та класу Main, а також налаштування pom.xml фреймворку Maven.

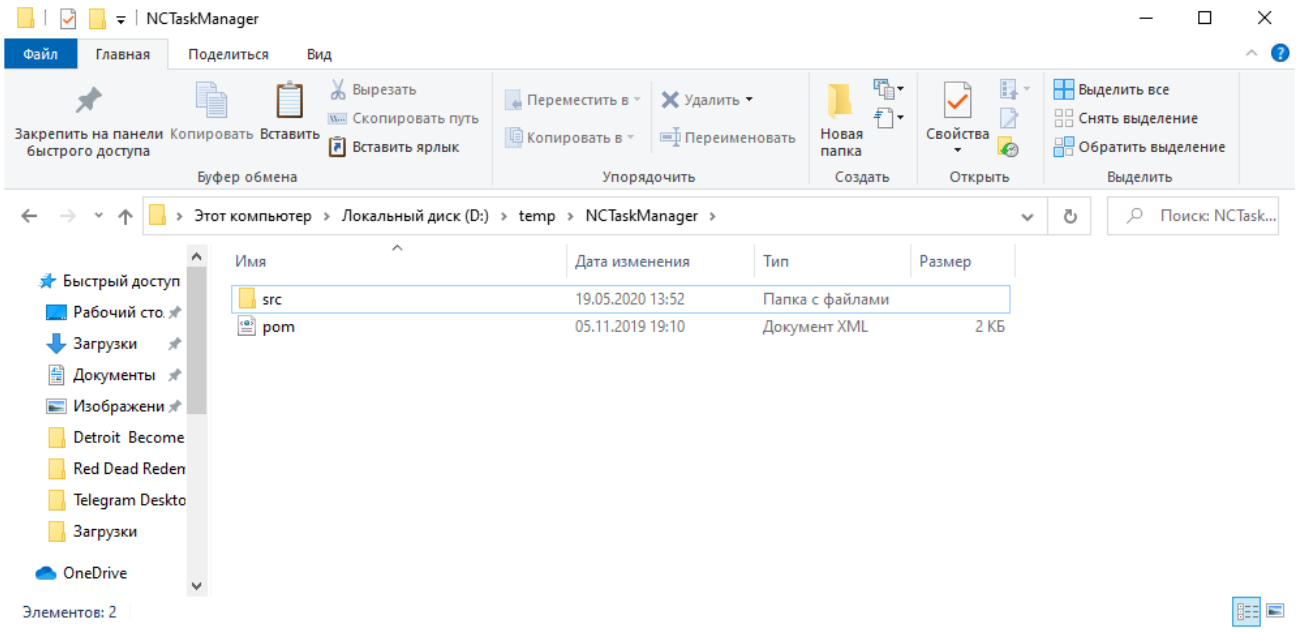

Рисунок 33 – Вміст архіву

Студент повинен розпакувати даний архів та відкрити проект у середовищі Intellij Idea.

Далі студент знаходить клас Main, та відкриває його. (Рисунок 34)

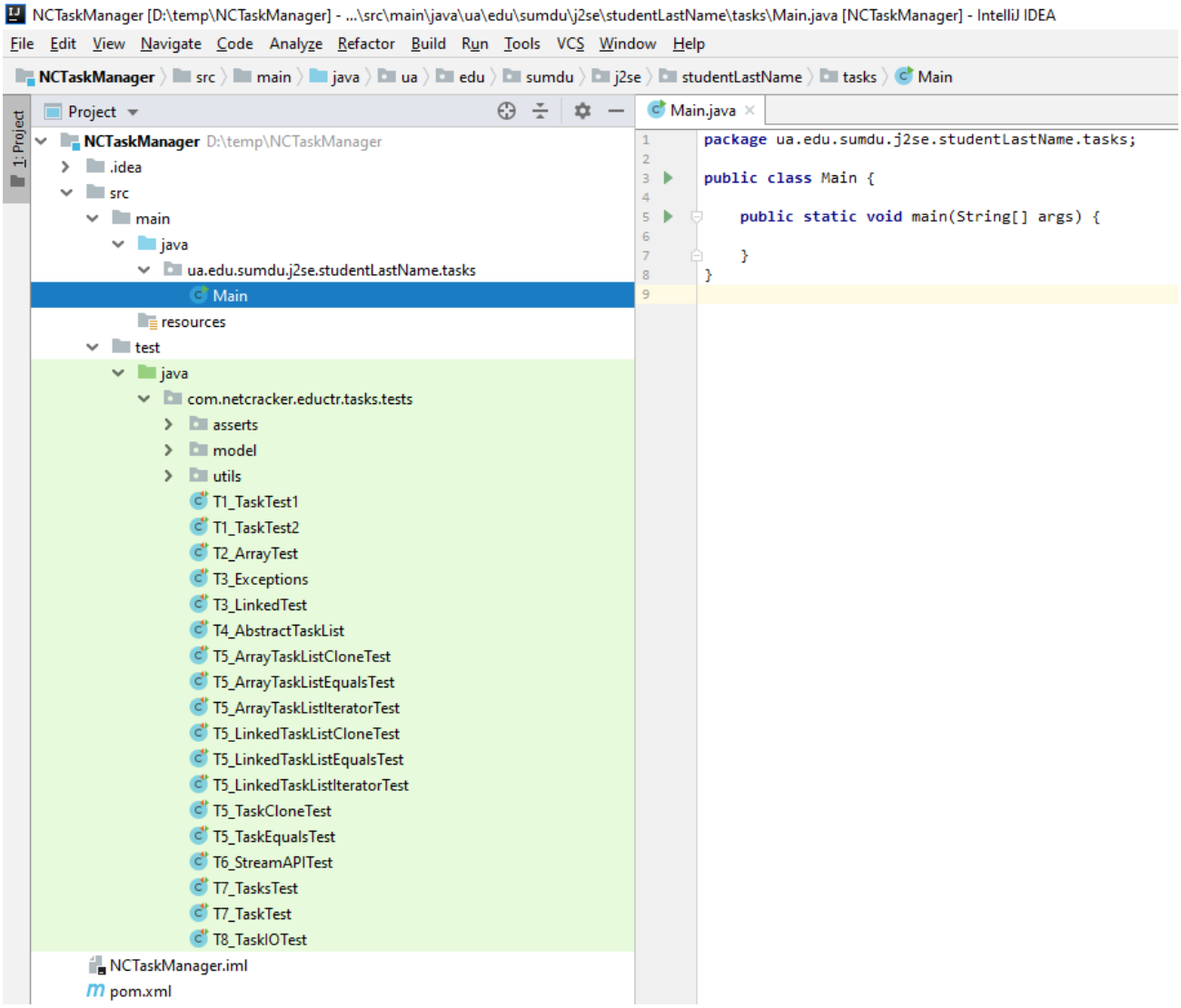

Рисунок 34 – Клас Main у структурі проекту

У пакеті класу Main студенту потрібно виділити studentLastName, натиснути правою копкою миші та обрати Refactor > Rename. (Рисунок 35)

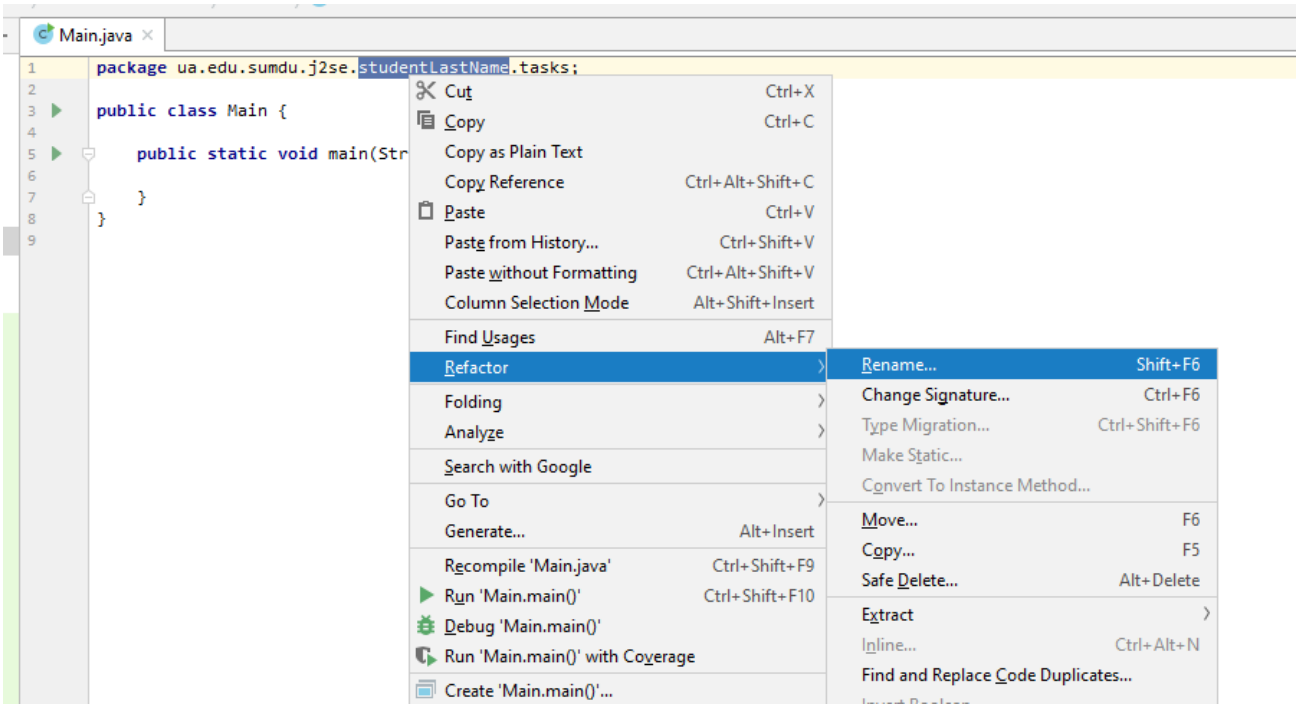

Рисунок 35 – Зміна назви пакету

У модальному вікні Rename студент повинен написати своє прізвище англійськими літерами та натиснути кнопку Refactor. (Рисунок 36)

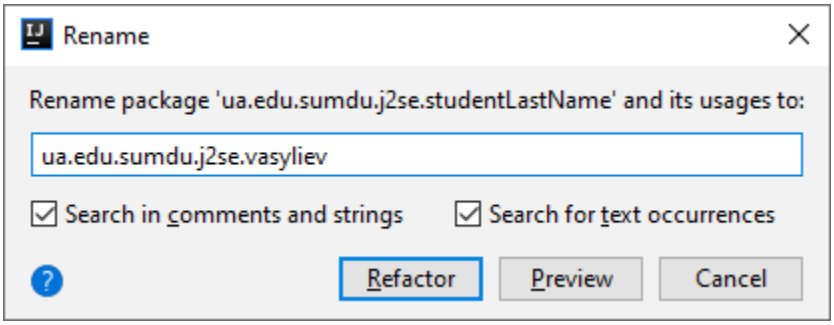

Рисунок 36 – Редагування назви пакету

Далі студент повинен у правому верхньому кутку натиснути кнопку Add Configuration. У відкритому вікні натиснути символ «+» та обрати JUnit. (Рисунок 37)

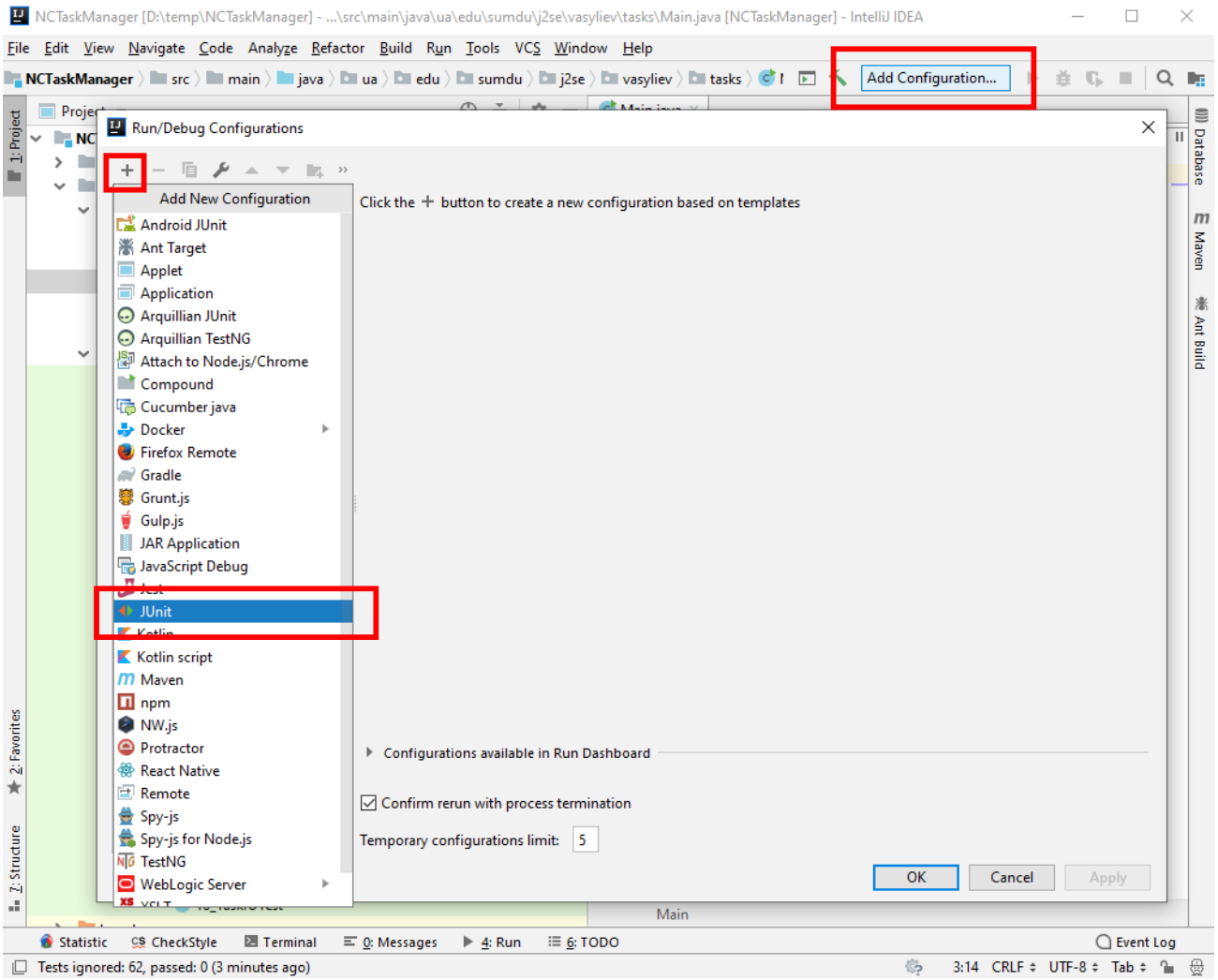

Рисунок 37 – Налаштування автотестів

У налаштуваннях справа ввести «Start Test» у поле Name. З випадаючого списку Tesk kind обрати «All in package». У поле Package ввести com.netcracker.eductr.tasks.tests. Натиснути Apply та OK. (Рисунок 38)

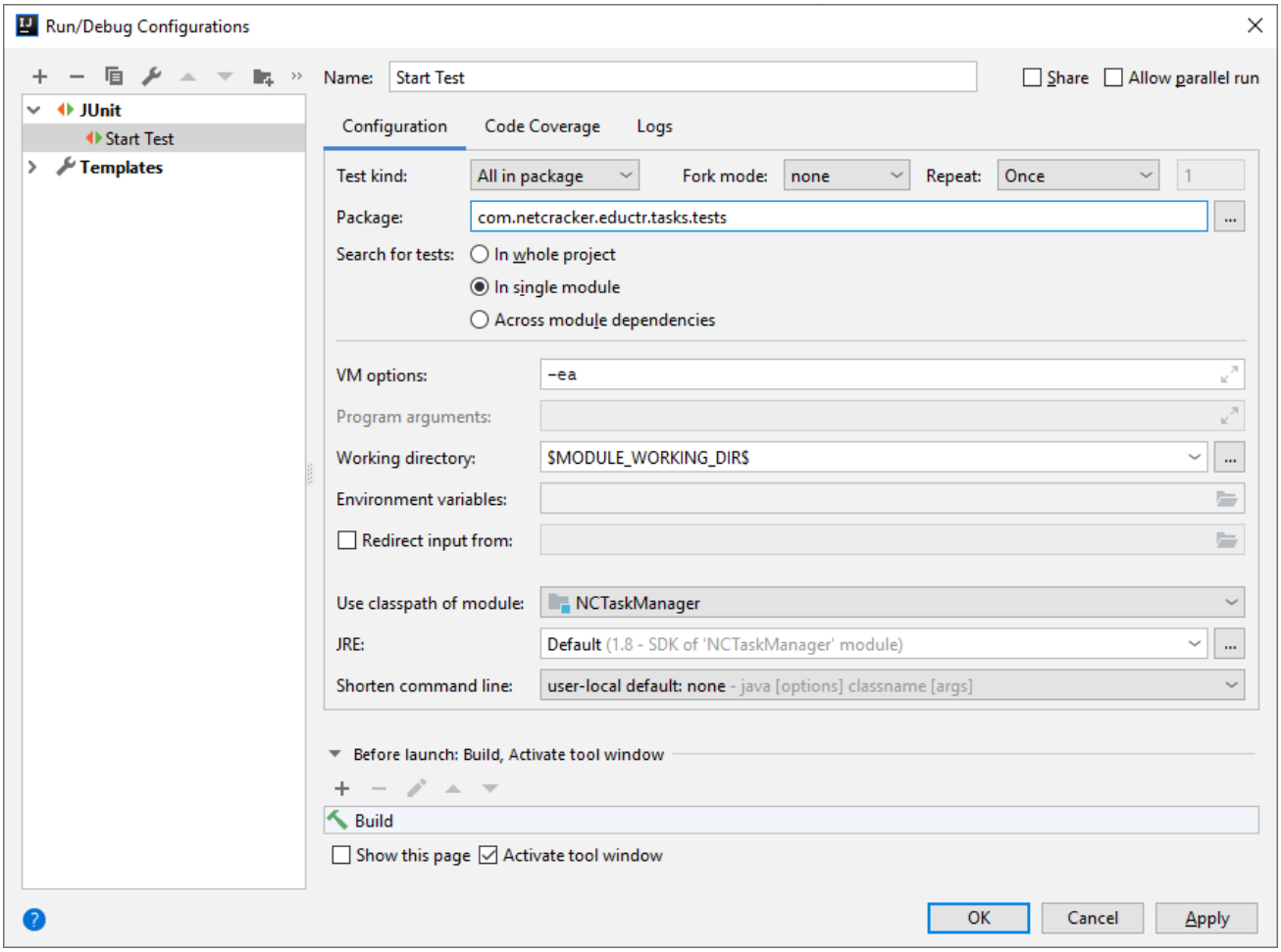

Рисунок 38 – Параметри автотестів

Після даних маніпуляцій проект є налаштованим і готовим для роботи.

Для першого завдання студенту необхідно створити клас Task у своєму пакеті з деякими полями та методами. Для демонстрації створимо пустий клас та запустимо тест. (Рисунок 39)

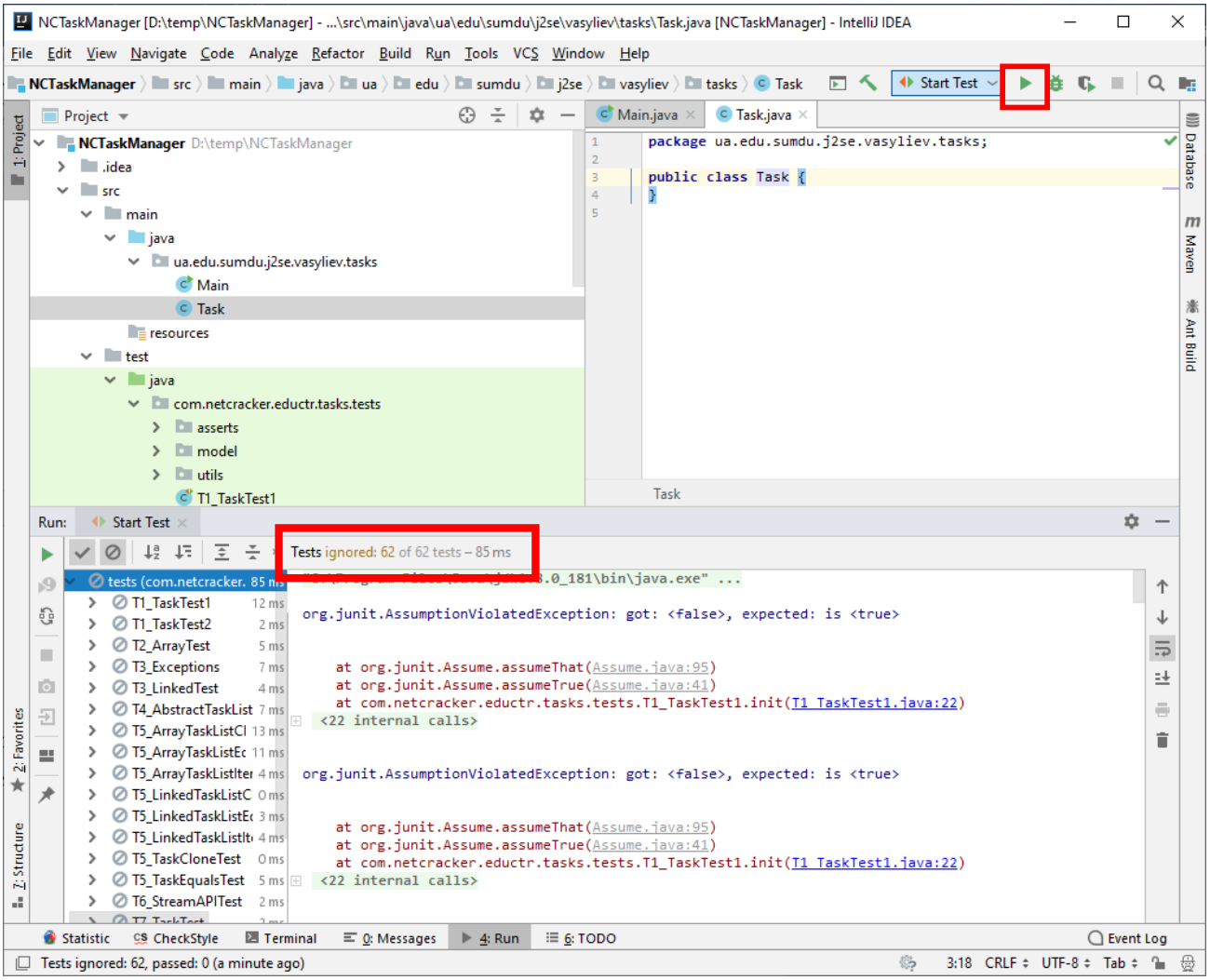

Рисунок 39 – Результат виконання тестів при наявному пустому класі

Як ми бачимо з результатів виконання автотестів, було проігноровано 62 тести. Це є очікуваним результатом, адже ми створили лише пустий клас у проекті та, фактично, нема що перевіряти. Якщо ми почнемо наповнювати класи логікою, то деякі з тестів вже почнуть виконуватись.

ВИСНОВКИ

У результаті виконання дипломної роботи було створено завдання для студентів курсу «Програмування мовою Java», а також програмне рішення (автотести) для перевірки правильності виконання завдань.

Було розглянуто існуючі системи оцінки знань на різних курсах програмування. На основі аналізу були створені завдання для НКЦ. Завдяки цим завданням студенти мають змогу отримати досвід роботи з Java 8, її новим функціоналом. Також студенти розглянуть збирання проекту за допомогою фреймворку Maven, також підключення сторонніх бібліотек до проекту. Створені завдання включають нові стандарти мови, які найбільш доцільно вивчати на стан 2020-го року.

Було розглянуто архітектуру проекту та принцип, за яким будуть створені оновлені автотести для перевірки правильності виконання завдань. Автотести були реалізовані на основі JUnit. Ключовим моментом реалізації оновлених тестів є те, що проект має змогу збиратись без єдиного класу для тестування. Цього вдалось досягти завдяки перевірці наявності необхідних класів шляхом рефлексії. Автотести інтегровані в середовище Intellij Idea для швидкої та зручної розробки програмних рішень мовою Java.

На час написання звіту даної роботи завдання та автотести до них були залучені у викладанні курсу «Програмування мовою Java» у 2019 році. За попередніми спостереженнями, студентам стало більш зручно виконувати завдання та працювати з кодом. Був зафіксований випадок, коли студент сам знайшов логічну проблему у коді тесту, виконуючи налагодження програми, що свідчить про підвищення ефективності праці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Netcracker Technology [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Netcracker_Technology.
2. IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/IntelliJ_IDEA.
3. Stepik [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Stepik
4. What is maven [Електронний ресурс] // Apache Maven Project – Режим доступу до ресурсу: https://maven.apache.org/what-is-maven.html.
5. Varanasi B. Introducing Maven / B. Varanasi, S. Belida., 2014. – 120 с.
6. Модульне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Модульне_тестування.
7. JUnit [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JUnit.
8. Thomas D. Pragmatic Unit Testing in Java 8 with JUnit / D. Thomas, A. Hunt, J. Langr., 2015. – 236 с.
9. Jackson B. R. Maven: The Definitive Guide / Brian R. Jackson., 2015. – 350 с.

ДОДАТОК А – Лістинг класів

ClassFinder.java

```
package com.netcracker.eductr.tasks.tests.utils;
import com.netcracker.eductr.tasks.tests.model.*;
import java.io.*;
import java.time.LocalDateTime;

import static com.netcracker.eductr.tasks.tests.model.BaseCreator.isOld;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.*;

public class ClassFinder {
   private static final String BASE_PACKAGE = "ua.edu.sumdu.j2se.studentLastName.tasks";

   public static boolean checkClassExistence(Types.classTypes type) {
      try {
         switch (type) {
            case TASK_BASE:
               Class<?> clazz = getTargetClass(type);
               Task.setTargetClass(clazz);
               boolean isOld = false;
               try {
                  isOld = clazz.getConstructor(String.class, int.class) != null;
               } catch (NoSuchMethodException e) {
               }
               BaseCreator.setIsOld(isOld);
               return true;

            case ARRAY_LIST:
            case LINKED_LIST:
            case ABSTRACT_LIST:
               TaskList.setTargetClass(getTargetClass(type));
               return true;

            case LIST_TYPES:
               ListTypes.setTargetClass(getTargetClass(type));
               return true;

            case TASK_LIST_FACTORY:
               TaskListFactory.setTargetClass(getTargetClass(type));
               return true;

            case TASKS:
               Tasks.setTargetClass(getTargetClass(type));
               return true;

            case TASK_IO:
               TaskIO.setTargetClass(getTargetClass(type));
               return true;

            default:
               return false;
```

```
         }
      } catch (ClassNotFoundException e) {
         return false;
      }
   }

   public static Class<?> getTargetClass(Types.classTypes clazz) throws
ClassNotFoundException {
      switch (clazz) {
         case TASK_BASE:
            return Class.forName(BASE_PACKAGE + ".Task");
         case ARRAY_LIST:
            return Class.forName(BASE_PACKAGE + ".ArrayTaskList");
         case LINKED_LIST:
            return Class.forName(BASE_PACKAGE + ".LinkedTaskList");
         case ABSTRACT_LIST:
            return Class.forName(BASE_PACKAGE + ".AbstractTaskList");
         case LIST_TYPES:
            return Class.forName(BASE_PACKAGE + ".ListTypes");
         case TASK_LIST_FACTORY:
            return Class.forName(BASE_PACKAGE + ".TaskListFactory");
         case TASKS:
            return Class.forName(BASE_PACKAGE + ".Tasks");
         case TASK_IO:
            return Class.forName(BASE_PACKAGE + ".TaskIO");
         default:
            throw new ClassNotFoundException();
      }
   }

   public static boolean checkMethodExistence(Types.methodTypes method, Types.classTypes
clazz) {
      try {
         switch (method) {
            case SIZE:
               return getTargetClass(clazz).getMethod("size") != null;
            case NEXT_TIME_AFTER:
               return (isOld()
                     ? getTargetClass(clazz).getMethod("nextTimeAfter", int.class)
                     : getTargetClass(clazz).getMethod("nextTimeAfter",
LocalDateTime.class)) != null;
            case ITERATOR:
               return getTargetClass(clazz).getMethod("iterator") != null;
            case CLONE:
               return getTargetClass(clazz).getMethod("clone") != null;
            case EQUALS:
               return getTargetClass(clazz).getMethod("equals", Object.class) != null;
            case HASH_CODE:
               return getTargetClass(clazz).getMethod("hashCode") != null;
            case GET_STREAM:
               return getTargetClass(clazz).getMethod("getStream") != null;
            case INCOMING:
               return getTargetClass(clazz).getMethod("incoming", Iterable.class,
LocalDateTime.class, LocalDateTime.class) != null;
            case READ_BINARY:
               return getTargetClass(clazz).getMethod("read",
getTargetClass(ABSTRACT_LIST), InputStream.class) != null
```

```
                     && getTargetClass(clazz).getMethod("readBinary",
getTargetClass(ABSTRACT_LIST), File.class) != null;
            case WRITE_BINARY:
               return getTargetClass(clazz).getMethod("write",
getTargetClass(ABSTRACT_LIST), OutputStream.class) != null
                     && getTargetClass(clazz).getMethod("writeBinary",
getTargetClass(ABSTRACT_LIST), File.class) != null;
            case READ:
               return getTargetClass(clazz).getMethod("read",
getTargetClass(ABSTRACT_LIST), Reader.class) != null
                     && getTargetClass(clazz).getMethod("readText",
getTargetClass(ABSTRACT_LIST), File.class) != null;
            case WRITE:
               return getTargetClass(clazz).getMethod("write",
getTargetClass(ABSTRACT_LIST), Writer.class) != null
                     && getTargetClass(clazz).getMethod("writeText",
getTargetClass(ABSTRACT_LIST), File.class) != null;
            case GET_TIME:
               return getTargetClass(clazz).getMethod("getTime") != null;
            case GET_START_TIME:
               return getTargetClass(clazz).getMethod("getStartTime") != null;
            case GET_END_TIME:
               return getTargetClass(clazz).getMethod("getEndTime") != null;
            case SET_TIME:
               return (isOld()
                     ? getTargetClass(clazz).getMethod("setTime", int.class)
                     : getTargetClass(clazz).getMethod("setTime", LocalDateTime.class))
!= null;
            case IS_ACTIVE:
               return getTargetClass(clazz).getMethod("isActive") != null;
            case IS_REPEATED:
               return getTargetClass(clazz).getMethod("isRepeated") != null;
            case GET_REPEAT_INTERVAL:
               return getTargetClass(clazz).getMethod("getRepeatInterval") != null;
            case GET_TITLE:
               return getTargetClass(clazz).getMethod("getTitle") != null;
            case ADD:
               return getTargetClass(clazz).getMethod("add", getTargetClass(TASK_BASE))
!= null;
            case GET_TASK:
               return getTargetClass(clazz).getMethod("getTask", int.class) != null;
         }

      } catch (ClassNotFoundException | NoSuchMethodException e) {
      }
      return false;
   }

   public static boolean checkDeclaredMethodExistence(Types.methodTypes method,
Types.classTypes clazz) {
      try {
         switch (method) {
            case SIZE:
               return getTargetClass(clazz).getDeclaredMethod("size") != null;
            case ADD:
               return getTargetClass(clazz).getDeclaredMethod("add",
getTargetClass(TASK_BASE)) != null;
```

```
            case REMOVE:
               return getTargetClass(clazz).getDeclaredMethod("remove",
getTargetClass(TASK_BASE)) != null;
            case GET_TASK:
               return getTargetClass(clazz).getDeclaredMethod("getTask", int.class) !=
null;
            case INCOMING:
               return getTargetClass(clazz).getDeclaredMethod("incoming", int.class,
int.class) != null;
            case EQUALS:
               return getTargetClass(clazz).getDeclaredMethod("equals", Object.class) !=
null;
            case HASH_CODE:
               return getTargetClass(clazz).getDeclaredMethod("hashCode") != null;
         }
      } catch (ClassNotFoundException | NoSuchMethodException e) {
      }
      return false;
   }

   public static boolean checkConstructorExistence(Types.constructorTypes constructor,
Types.classTypes clazz) {
      try {
         switch (constructor) {
            case TITLE_TIME:
               return (isOld()
                     ? getTargetClass(clazz).getConstructor(String.class, int.class)
                     : getTargetClass(clazz).getConstructor(String.class,
LocalDateTime.class)) != null;
            case TITLE_START_END_INTERVAL:
               return (isOld()
                     ? getTargetClass(clazz).getConstructor(String.class, int.class,
int.class, int.class)
                     : getTargetClass(clazz).getConstructor(String.class,
LocalDateTime.class, LocalDateTime.class, int.class)) != null;
         }
      } catch (NoSuchMethodException | ClassNotFoundException e) {
      }
      return false;
   }
}
```

Types.java

```
package com.netcracker.eductr.tasks.tests.utils;
public class Types {
    public enum classTypes {
        TASK_BASE,
        ARRAY_LIST,
        LINKED_LIST,
        ABSTRACT_LIST,
        LIST_TYPES,
        TASK_LIST_FACTORY,
        TASKS,
```

```
        TASK_IO
    }
    public enum methodTypes {
        NEXT_TIME_AFTER,
        SIZE,
        ADD,
        REMOVE,
        GET_TASK,
        INCOMING,
        ITERATOR,
        CLONE,
        EQUALS,
        HASH_CODE,
        GET_STREAM,
        WRITE_BINARY,
        READ_BINARY,
        WRITE,
        READ,
        GET_TITLE,
        IS_REPEATED,
        GET_TIME,
        IS_ACTIVE,
        GET_START_TIME,
        GET_END_TIME,
        GET_REPEAT_INTERVAL,
        SET_TIME
    }
    public enum constructorTypes {
        TITLE_TIME,
        TITLE_START_END_INTERVAL
    }
}
```

Task.java

```
package com.netcracker.eductr.tasks.tests.model;

import java.lang.reflect.InvocationTargetException;
import java.time.LocalDateTime;

public class Task extends BaseObject {
    private static Class<?> targetClass;

    public static void setTargetClass(Class<?> targetClass) {
        Task.targetClass = targetClass;
    }

    public Task(String title, int time) {
        try {
            instance = targetClass.getConstructor(String.class,
int.class).newInstance(title, time);
        } catch (InstantiationException | IllegalAccessException |
InvocationTargetException | NoSuchMethodException e) {
        }
```

```
    }

    public Task(Object instance) {
        this.instance = instance;
    }

    public String getTitle() {
        try {
            return (String) targetClass.getMethod("getTitle").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public void setTitle(String title) {
        try {
            targetClass.getMethod("setTitle", String.class).invoke(instance, title);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {

        }
    }

    public boolean isRepeated() {
        try {
            return (boolean) targetClass.getMethod("isRepeated").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return false;
        }
    }

    public boolean isActive() {
        try {
            return (boolean) targetClass.getMethod("isActive").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return false;
        }
    }

    public Object getTime() {
        try {
            return targetClass.getMethod("getTime").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public Object getStartTime() {
        try {
            return targetClass.getMethod("getStartTime").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
```

```
        }
    }

    public Object getEndTime() {
        try {
            return targetClass.getMethod("getEndTime").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public int getRepeatInterval() {
        try {
            return (int) targetClass.getMethod("getRepeatInterval").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return Integer.MIN_VALUE;
        }
    }

    public Task(String title, int start, int end, int interval) {
        try {
            instance = targetClass.getConstructor(String.class, int.class, int.class,
int.class).newInstance(title, start, end, interval);
        } catch (InstantiationException | IllegalAccessException |
InvocationTargetException | NoSuchMethodException e) {

        }
    }

    public void setTime(int time) {
        try {
            targetClass.getMethod("setTime", int.class).invoke(instance, time);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {

        }
    }

    public void setTime(int start, int end, int interval) {
        try {
            targetClass.getMethod("setTime", int.class, int.class,
int.class).invoke(instance, start, end, interval);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {

        }
    }

    public void setActive(boolean active) {
        try {
            targetClass.getMethod("setActive", boolean.class).invoke(instance, active);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {

        }
```

```
    }

    public int nextTimeAfter(int time) {
        try {
            return (int) targetClass.getMethod("nextTimeAfter",
int.class).invoke(instance, time);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return Integer.MIN_VALUE;
        }
    }

    public static Class<?> getTargetClass() {
        return targetClass;
    }

    public static void createWithException(String title, int time) throws Throwable {
        try {
            targetClass.getConstructor(String.class, int.class).newInstance(title,
time);
        } catch (InstantiationException | IllegalAccessException |
InvocationTargetException | NoSuchMethodException e) {
            throw e.getCause();
        }
    }

    public Task clone() {
        try {
            return new Task(targetClass.getMethod("clone").invoke(instance));
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public boolean equals(Object o) {
        try {
            return (boolean) targetClass.getMethod("equals",
Object.class).invoke(instance, o instanceof Task ? ((Task) o).instance : o);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return false;
        }
    }

    public int hashCode() {
        try {
            return (int) targetClass.getMethod("hashCode").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return Integer.MIN_VALUE;
        }
    }

    public Task(String title, LocalDateTime time) {
        try {
            instance = targetClass.getConstructor(String.class,
```

```
LocalDateTime.class).newInstance(title, time);
        } catch (InstantiationException | IllegalAccessException |
InvocationTargetException | NoSuchMethodException e) {
        }
    }

    public void setTime(LocalDateTime time) {
        try {
            targetClass.getMethod("setTime", LocalDateTime.class).invoke(instance,
time);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {

        }
    }

    public Task(String title, LocalDateTime start, LocalDateTime end, int interval) {
        try {
            instance = targetClass.getConstructor(String.class, LocalDateTime.class,
LocalDateTime.class, int.class).newInstance(title, start, end, interval);
        } catch (InstantiationException | IllegalAccessException |
InvocationTargetException | NoSuchMethodException e) {

        }
    }

    public void setTime(LocalDateTime start, LocalDateTime end, int interval) {
        try {
            targetClass.getMethod("setTime", LocalDateTime.class, LocalDateTime.class,
int.class).invoke(instance, start, end, interval);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {

        }
    }

    public LocalDateTime nextTimeAfter(LocalDateTime time) {
        try {
            return (LocalDateTime) targetClass.getMethod("nextTimeAfter",
LocalDateTime.class).invoke(instance, time);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return LocalDateTime.MIN;
        }
    }

    public static void createWithException(String title, LocalDateTime time) throws
Throwable {
        try {
            targetClass.getConstructor(String.class,
LocalDateTime.class).newInstance(title, time);
        } catch (InstantiationException | IllegalAccessException |
InvocationTargetException | NoSuchMethodException e) {
            throw e.getCause();
        }}
}
```

BaseObejct.java

```
package com.netcracker.eductr.tasks.tests.model;

public abstract class BaseObject {
    protected Object instance;

    public Object getInstance() {
        return instance;
    }
}
```

T1_TaskTest1.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.Task;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.BaseCreator.isOld;
import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.*;
import static com.netcracker.eductr.tasks.tests.utils.DatesTimes.*;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.TASK_BASE;
import static
com.netcracker.eductr.tasks.tests.utils.Types.constructorTypes.TITLE_START_END_INTERVAL;
import static com.netcracker.eductr.tasks.tests.utils.Types.constructorTypes.TITLE_TIME;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.*;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T1_TaskTest1 {

   @BeforeClass
   public static void init() {
      Assume.assumeTrue(checkClassExistence(TASK_BASE));
      Assume.assumeTrue(checkConstructorExistence(TITLE_TIME, TASK_BASE));
      Assume.assumeTrue(checkConstructorExistence(TITLE_START_END_INTERVAL, TASK_BASE));
      Assume.assumeTrue(checkMethodExistence(GET_TITLE, TASK_BASE));
      Assume.assumeTrue(checkMethodExistence(IS_REPEATED, TASK_BASE));
      Assume.assumeTrue(checkMethodExistence(IS_ACTIVE, TASK_BASE));
      Assume.assumeTrue(checkMethodExistence(GET_TIME, TASK_BASE));
      Assume.assumeTrue(checkMethodExistence(GET_START_TIME, TASK_BASE));
      Assume.assumeTrue(checkMethodExistence(GET_END_TIME, TASK_BASE));
      Assume.assumeTrue(checkMethodExistence(GET_REPEAT_INTERVAL, TASK_BASE));
      Assume.assumeTrue(checkMethodExistence(SET_TIME, TASK_BASE));
   }

   @Test
   public void part1_testConstructorNonRepeat() {
      if (isOld()) {
         Task task = create("Task title", 42);
```

```
         Assert.assertEquals("Задача має неочікувану назву", "Task title",
task.getTitle());

         Assert.assertFalse("Задача не повинна повторюватись", task.isRepeated());

         Assert.assertFalse("Задача повинна бути неактивною", task.isActive());

         Assert.assertEquals("Задача має неочікуваний час виконання", 42,
task.getTime());

         Assert.assertEquals("Задача має неочікуваний час початку", 42,
task.getStartTime());

         Assert.assertEquals("Задача має неочікуваний час закінчення", 42,
task.getEndTime());

         Assert.assertEquals("Задача має неочікуваний інтервал повторення", 0,
task.getRepeatInterval());
      } else {
         Task task = create("Task title", FROM_NOW_42);

         Assert.assertEquals("Задача має неочікувану назву", "Task title",
task.getTitle());

         Assert.assertFalse("Задача не повинна повторюватись", task.isRepeated());

         Assert.assertFalse("Задача повинна бути неактивною", task.isActive());

         Assert.assertEquals("Задача має неочікуваний час виконання", FROM_NOW_42,
task.getTime());

         Assert.assertEquals("Задача має неочікуваний час початку", FROM_NOW_42,
task.getStartTime());

         Assert.assertEquals("Задача має неочікуваний час закінчення", FROM_NOW_42,
task.getEndTime());

         Assert.assertEquals("Задача має неочікуваний інтервал повторення", 0,
task.getRepeatInterval());
      }
   }

   @Test
   public void part2_testConstructorRepeat() {
      if (isOld()) {
         Task task = create("Task title", 5, 25, 3);

         Assert.assertEquals("Задача має неочікувану назву", "Task title",
task.getTitle());

         Assert.assertTrue("Задача повинна повторюватись", task.isRepeated());

         Assert.assertFalse("Задача повинна бути неактивною", task.isActive());

         Assert.assertEquals("Задача має неочікуваний час початку", 5,
task.getStartTime());
```

```
         Assert.assertEquals("Задача має неочікуваний час закінчення", 25,
task.getEndTime());

         Assert.assertEquals("Задача має неочікуваний інтервал повторення", 3,
task.getRepeatInterval());

         Assert.assertEquals("Задача має неочікуваний час виконання", 5,
task.getTime());
      } else {
         Task task = create("Task title", FROM_NOW_5, FROM_NOW_25, 3);

         Assert.assertEquals("Задача має неочікувану назву", "Task title",
task.getTitle());

         Assert.assertTrue("Задача повинна повторюватись", task.isRepeated());

         Assert.assertFalse("Задача повинна бути неактивною", task.isActive());

         Assert.assertEquals("Задача має неочікуваний час початку", FROM_NOW_5,
task.getStartTime());

         Assert.assertEquals("Задача має неочікуваний час закінчення", FROM_NOW_25,
task.getEndTime());

         Assert.assertEquals("Задача має неочікуваний інтервал повторення", 3,
task.getRepeatInterval());

         Assert.assertEquals("Задача має неочікуваний час виконання", FROM_NOW_5,
task.getTime());
      }
   }

   @Test
   public void part3_testSetTime()
   {
      if (isOld()) {
         Task task = create("Task title", 5, 25, 3);

         task.setTime(42);

         Assert.assertFalse("Задача не повинна повторюватись", task.isRepeated());

         Assert.assertEquals("Задача має неочікуваний час виконання", 42,
task.getTime());

         Assert.assertEquals("Задача має неочікуваний час початку", 42,
task.getStartTime());

         Assert.assertEquals("Задача має неочікуваний час закінчення", 42,
task.getEndTime());

         Assert.assertEquals("Задача має неочікуваний інтервал повторення", 0,
task.getRepeatInterval());

         task.setTime(5, 25, 3);

         Assert.assertTrue("Задача повинна повторюватись", task.isRepeated());
```

```
        Assert.assertEquals("Задача має неочікуваний час початку", 5,
task.getStartTime());

        Assert.assertEquals("Задача має неочікуваний час закінчення", 25,
task.getEndTime());

        Assert.assertEquals("Задача має неочікуваний інтервал повторення", 3,
task.getRepeatInterval());

        Assert.assertEquals("Задача має неочікуваний час виконання", 5,
task.getTime());
      } else {
        Task task = create("Task title", FROM_NOW_5, FROM_NOW_25, 3);

        task.setTime(FROM_NOW_42);

        Assert.assertFalse("Задача не повинна повторюватись", task.isRepeated());

        Assert.assertEquals("Задача має неочікуваний час виконання", FROM_NOW_42,
task.getTime());

        Assert.assertEquals("Задача має неочікуваний час початку", FROM_NOW_42,
task.getStartTime());

        Assert.assertEquals("Задача має неочікуваний час закінчення", FROM_NOW_42,
task.getEndTime());

        Assert.assertEquals("Задача має неочікуваний інтервал повторення", 0,
task.getRepeatInterval());

        task.setTime(FROM_NOW_5, FROM_NOW_25, 3);

        Assert.assertTrue("Задача повинна повторюватись", task.isRepeated());

        Assert.assertEquals("Задача має неочікуваний час початку", FROM_NOW_5,
task.getStartTime());

        Assert.assertEquals("Задача має неочікуваний час закінчення", FROM_NOW_25,
task.getEndTime());

        Assert.assertEquals("Задача має неочікуваний інтервал повторення", 3,
task.getRepeatInterval());

        Assert.assertEquals("Задача має неочікуваний час виконання", FROM_NOW_5,
task.getTime());
      }
   }
}
```

TaskList.java

```
package com.netcracker.eductr.tasks.tests.model;

import java.lang.reflect.InvocationTargetException;
import java.time.LocalDateTime;
import java.util.Iterator;
import java.util.stream.Stream;

public class TaskList extends BaseObject {
    private static Class<?> targetClass;

    public static void setTargetClass(Class<?> targetClass) {
        TaskList.targetClass = targetClass;
    }

    public TaskList(Object instance) {
        this.instance = instance;
    }

    public static Class<?> getTargetClass() {
        return targetClass;
    }

    public TaskList() {
        try {
            instance = targetClass.newInstance();
        } catch (InstantiationException | IllegalAccessException e) {
            e.printStackTrace();
        }
    }

    public int size() {
        try {
            return (int) targetClass.getMethod("size").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return Integer.MAX_VALUE;
        }
    }

    public void add(Task task) {
        try {
            targetClass.getMethod("add", Task.getTargetClass()).invoke(instance,
task.getInstance());
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {

        }
    }

    public Task getTask(int index) {
        try {
            return new Task(targetClass.getMethod("getTask", int.class).invoke(instance,
index));
```

```
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public boolean remove(Task task) {
        try {
            return (boolean) targetClass.getMethod("remove",
Task.getTargetClass()).invoke(instance, task.getInstance());
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return false;
        }
    }

    public TaskList incoming(int from, int to) {
        try {
            Object o = targetClass.getMethod("incoming", int.class,
int.class).invoke(instance, from, to);
            targetClass = o.getClass();
            return new TaskList(o);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public void getTaskWithException(int index) throws Throwable {
        try {
            new Task(targetClass.getMethod("getTask", int.class).invoke(instance,
index));
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            throw e.getCause();
        }
    }

    public Iterator<Task> iterator() {
        try {
            return (Iterator<Task>) targetClass.getMethod("iterator").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public TaskList clone() {
        try {
            return new TaskList(targetClass.getMethod("clone").invoke(instance));
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }
```

```
    public boolean equals(Object o) {
        try {
            return (boolean) targetClass.getMethod("equals",
Object.class).invoke(instance, o instanceof TaskList ? ((TaskList) o).instance : o);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return false;
        }
    }

    public int hashCode() {
        try {
            return (int) targetClass.getMethod("hashCode").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return Integer.MIN_VALUE;
        }
    }

    public Stream<Task> getStream() {
        try {
            return (Stream<Task>) targetClass.getMethod("getStream").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }
    public TaskList incoming(LocalDateTime from, LocalDateTime to) {
        try {
            Object o = targetClass.getMethod("incoming", LocalDateTime.class,
LocalDateTime.class).invoke(instance, from, to);
            targetClass = o.getClass();
            return new TaskList(o);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }
}
```

BaseCreator.java

```
package com.netcracker.eductr.tasks.tests.model;

public class BaseCreator {
  protected static boolean IS_OLD;

  public static void setIsOld(boolean isOld) {
    IS_OLD = isOld;
  }

  public static boolean isOld() {
    return IS_OLD;
  }}
```

ListTypes.java

```
package com.netcracker.eductr.tasks.tests.model;

import java.util.Arrays;
import java.util.List;

public class ListTypes {
    private static Class<?> targetClass;
    private static List<String> set = Arrays.asList("ARRAY", "LINKED");

    public static void setTargetClass(Class<?> targetClass) {
        ListTypes.targetClass = targetClass;
    }

    public static int count() {
        return targetClass.getClasses()[0].getFields().length;
    }

    public static boolean isRightNames() {
        return Arrays.stream(targetClass.getClasses()[0].getFields()).map(f ->
f.getName()).filter(n -> !set.contains(n)).count() == 0;
    }

    public static Class<?> getEnumClass() {
        return targetClass.getClasses()[0];
    }
}
```

TaskCreator.java

```
package com.netcracker.eductr.tasks.tests.model;

import java.time.LocalDateTime;

import static com.netcracker.eductr.tasks.tests.utils.DatesTimes.FROM_NOW_10;
import static com.netcracker.eductr.tasks.tests.utils.DatesTimes.NOW;

public class TaskCreator extends BaseCreator {
    public static Task create(String title) {
        return IS_OLD ? create(title, 5) : create(title, NOW);
    }

    public static Task create(String title, int time, boolean active) {
        Task task = create(title, time);
        task.setActive(active);
        return task;
    }

    public static Task create(String title, int time) {
        return new Task(title, time);
    }
```

```
    public static Task create(String title, int start, int end, int interval) {
        return new Task(title, start, end, interval);
    }

    public static Task create(String title, int start, int end, int interval, boolean
active) {
        Task task = create(title, start, end, interval);
        task.setActive(active);
        return task;
    }

    public static Task create(String title, LocalDateTime start, LocalDateTime end, int
interval) {
        return new Task(title, start, end, interval);
    }

    public static Task create(String title, LocalDateTime dateTime) {
        return new Task(title, dateTime);
    }

    public static Task create(String title, LocalDateTime dateTime, boolean active) {
        Task task = create(title, dateTime);
        task.setActive(active);
        return task;
    }

    public static Task create(String title, LocalDateTime start, LocalDateTime end, int
interval, boolean active) {
        Task task = create(title, start, end, interval);
        task.setActive(active);
        return task;
    }

    public static Task createA() {
        return IS_OLD ? create("A", 10) : create("A", NOW);
    }

    public static Task createB() {
        return IS_OLD ? create("B", 20) : create("B", FROM_NOW_10);
    }

    public static void modify(Task t) {
        t.setTitle("Some task");
        if (IS_OLD) t.setTime(42);
        else t.setTime(NOW);
        t.setActive(true);
    }
}
```

TaskIO.java

```
package com.netcracker.eductr.tasks.tests.model;

import java.io.*;
import java.lang.reflect.InvocationTargetException;

import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.getTargetClass;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.ABSTRACT_LIST;

public class TaskIO extends BaseObject {
  private static Class<?> targetClass;

  public static void setTargetClass(Class<?> targetClass) {
    TaskIO.targetClass = targetClass;
  }

  public static void write(TaskList tasks, OutputStream out) {
    try {
      targetClass.getMethod("write", getTargetClass(ABSTRACT_LIST),
OutputStream.class).invoke(null, tasks.getInstance(), out);
    } catch (IllegalAccessException | InvocationTargetException | NoSuchMethodException
| ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
  public static void read(TaskList tasks, InputStream in) {
    try {
      targetClass.getMethod("read", getTargetClass(ABSTRACT_LIST),
InputStream.class).invoke(null, tasks.getInstance(), in);
    } catch (IllegalAccessException | InvocationTargetException | NoSuchMethodException
| ClassNotFoundException e) {
      e.printStackTrace();
    }
  }

  public static void writeBinary(TaskList tasks, File file) {
    try {
      targetClass.getMethod("writeBinary", getTargetClass(ABSTRACT_LIST),
File.class).invoke(null, tasks.getInstance(), file);
    } catch (IllegalAccessException | InvocationTargetException | NoSuchMethodException
| ClassNotFoundException e) {
      e.printStackTrace();
    }
  }

  public static void readBinary(TaskList tasks, File file) {
    try {
      targetClass.getMethod("readBinary", getTargetClass(ABSTRACT_LIST),
File.class).invoke(null, tasks.getInstance(), file);
    } catch (IllegalAccessException | InvocationTargetException | NoSuchMethodException
| ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
```

```
  public static void write(TaskList tasks, Writer out) {
    try {
      targetClass.getMethod("write", getTargetClass(ABSTRACT_LIST),
Writer.class).invoke(null, tasks.getInstance(), out);
    } catch (IllegalAccessException | InvocationTargetException | NoSuchMethodException
| ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
  public static void read(TaskList tasks, Reader in) {
    try {
      targetClass.getMethod("read", getTargetClass(ABSTRACT_LIST),
Reader.class).invoke(null, tasks.getInstance(), in);
    } catch (IllegalAccessException | InvocationTargetException | NoSuchMethodException
| ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
  public static void writeText(TaskList tasks, File file) {
    try {
      targetClass.getMethod("writeText", getTargetClass(ABSTRACT_LIST),
File.class).invoke(null, tasks.getInstance(), file);
    } catch (IllegalAccessException | InvocationTargetException | NoSuchMethodException
| ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
  public static void readText(TaskList tasks, File file) {
    try {
      targetClass.getMethod("readText", getTargetClass(ABSTRACT_LIST),
File.class).invoke(null, tasks.getInstance(), file);
    } catch (IllegalAccessException | InvocationTargetException | NoSuchMethodException
| ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
}
```

TaskList.java

```
package com.netcracker.eductr.tasks.tests.model;

import java.lang.reflect.InvocationTargetException;
import java.time.LocalDateTime;
import java.util.Iterator;
import java.util.stream.Stream;

public class TaskList extends BaseObject {
    private static Class<?> targetClass;

    public static void setTargetClass(Class<?> targetClass) {
        TaskList.targetClass = targetClass;
    }
```

```
    public TaskList(Object instance) {
        this.instance = instance;
    }

    public static Class<?> getTargetClass() {
        return targetClass;
    }

    public TaskList() {
        try {
            instance = targetClass.newInstance();
        } catch (InstantiationException | IllegalAccessException e) {
            e.printStackTrace();
        }
    }

    public int size() {
        try {
            return (int) targetClass.getMethod("size").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return Integer.MAX_VALUE;
        }
    }

    public void add(Task task) {
        try {
            targetClass.getMethod("add", Task.getTargetClass()).invoke(instance,
task.getInstance());
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {

        }
    }

    public Task getTask(int index) {
        try {
            return new Task(targetClass.getMethod("getTask", int.class).invoke(instance,
index));
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public boolean remove(Task task) {
        try {
            return (boolean) targetClass.getMethod("remove",
Task.getTargetClass()).invoke(instance, task.getInstance());
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return false;
        }
    }

    public TaskList incoming(int from, int to) {
```

```
        try {
            Object o = targetClass.getMethod("incoming", int.class,
int.class).invoke(instance, from, to);
            targetClass = o.getClass();
            return new TaskList(o);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public void getTaskWithException(int index) throws Throwable {
        try {
            new Task(targetClass.getMethod("getTask", int.class).invoke(instance,
index));
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            throw e.getCause();
        }
    }

    public Iterator<Task> iterator() {
        try {
            return (Iterator<Task>) targetClass.getMethod("iterator").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public TaskList clone() {
        try {
            return new TaskList(targetClass.getMethod("clone").invoke(instance));
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public boolean equals(Object o) {
        try {
            return (boolean) targetClass.getMethod("equals",
Object.class).invoke(instance, o instanceof TaskList ? ((TaskList) o).instance : o);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return false;
        }
    }

    public int hashCode() {
        try {
            return (int) targetClass.getMethod("hashCode").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return Integer.MIN_VALUE;
        }
```

```
    }

    public Stream<Task> getStream() {
        try {
            return (Stream<Task>) targetClass.getMethod("getStream").invoke(instance);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }
    public TaskList incoming(LocalDateTime from, LocalDateTime to) {
        try {
            Object o = targetClass.getMethod("incoming", LocalDateTime.class,
LocalDateTime.class).invoke(instance, from, to);
            targetClass = o.getClass();
            return new TaskList(o);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }
}
```

TaskListCreator.java

```
package com.netcracker.eductr.tasks.tests.model;
import java.util.stream.IntStream;
import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;
import static com.netcracker.eductr.tasks.tests.utils.DatesTimes.TODAY;
import static com.netcracker.eductr.tasks.tests.utils.DatesTimes.TOMORROW;

public class TaskListCreator extends BaseCreator {

  public static TaskList createA() {
    TaskList list = new TaskList();
    Task A = TaskCreator.createA();
    IntStream.range(0, 10).forEach(i -> list.add(A));
    return list;
  }

  public static TaskList createB() {
    TaskList list = new TaskList();
    Task B = TaskCreator.createB();
    IntStream.range(0, 10).forEach(i -> list.add(B));
    return list;
  }

  public static void modify(TaskList list) {
    IntStream.range(0, list.size()).mapToObj(i ->
list.getTask(0)).forEach(list::remove);
    Task A = TaskCreator.createA();
    IntStream.range(0, 10).forEach(i -> list.add(A));
  }
  public static TaskList createList() {
```

```
    TaskList tasks = new TaskList();
    tasks.add(create("A", TODAY));
    tasks.add(create("B", TODAY, TOMORROW, 3600));
    Task t = create("C", TODAY);
    t.setActive(true);
    tasks.add(t);
    t = create("B", TODAY, TOMORROW, 3600);
    t.setActive(true);
    tasks.add(t);
    return tasks;
  }
}
```

TaskListFactory.java

```
package com.netcracker.eductr.tasks.tests.model;

import com.netcracker.eductr.tasks.tests.utils.Types;

import java.lang.reflect.InvocationTargetException;
import java.util.Arrays;

import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.ARRAY_LIST;

public class TaskListFactory {
    private static Class<?> targetClass;

    public static void setTargetClass(Class<?> targetClass) {
        TaskListFactory.targetClass = targetClass;
    }

    public static TaskList createTaskList(Types.classTypes type) {
        try {
            Class<?> enumClass = ListTypes.getEnumClass();
            String target = type == ARRAY_LIST ? "ARRAY" : "LINKED";
            Object param = Arrays.stream(enumClass.getFields()).filter(f ->
target.equals(f.getName())).findFirst().get().get(null);
            return new TaskList(targetClass.getMethod("createTaskList",
enumClass).invoke(null, param));
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }
}
```

Tasks.java

```
package com.netcracker.eductr.tasks.tests.model;

import java.lang.reflect.InvocationTargetException;
import java.time.LocalDateTime;
import java.util.*;

public class Tasks extends BaseObject {
    private static Class<?> targetClass;

    public static void setTargetClass(Class<?> targetClass) {
        Tasks.targetClass = targetClass;
    }

    public static Iterable<Task> incoming(Iterable<Task> tasks, LocalDateTime start,
LocalDateTime end) {
        try {
            List listParam = new ArrayList();
            tasks.forEach(listParam::add);
            return (Iterable<Task>) targetClass.getMethod("incoming", Iterable.class,
LocalDateTime.class, LocalDateTime.class).invoke(null, listParam, start, end);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }

    public static SortedMap<LocalDateTime, Set<Task>> calendar(Iterable<Task> tasks,
LocalDateTime start, LocalDateTime end) {
        List listParam = new ArrayList();
        tasks.forEach(listParam::add);
        try {
            return (SortedMap<LocalDateTime, Set<Task>>)
targetClass.getMethod("calendar", Iterable.class, LocalDateTime.class,
LocalDateTime.class).invoke(null, listParam, start, end);
        } catch (IllegalAccessException | InvocationTargetException |
NoSuchMethodException e) {
            return null;
        }
    }
}
```

DatesTimes.java

```
package com.netcracker.eductr.tasks.tests.utils;

import java.time.LocalDateTime;

public class DatesTimes {
    public static final LocalDateTime NOW = LocalDateTime.now();
    public static final LocalDateTime FROM_NOW_1 = NOW.plusSeconds(1);
    public static final LocalDateTime FROM_NOW_3 = NOW.plusSeconds(3);
```

```
    public static final LocalDateTime FROM_NOW_5 = NOW.plusSeconds(5);
    public static final LocalDateTime FROM_NOW_9 = NOW.plusSeconds(9);
    public static final LocalDateTime FROM_NOW_10 = NOW.plusSeconds(10);
    public static final LocalDateTime FROM_NOW_25 = NOW.plusSeconds(25);
    public static final LocalDateTime FROM_NOW_30 = NOW.plusSeconds(30);
    public static final LocalDateTime FROM_NOW_40 = NOW.plusSeconds(40);
    public static final LocalDateTime FROM_NOW_42 = NOW.plusSeconds(42);
    public static final LocalDateTime FROM_NOW_50 = NOW.plusSeconds(50);
    public static final LocalDateTime FROM_NOW_51 = NOW.plusSeconds(51);
    public static final LocalDateTime FROM_NOW_55 = NOW.plusSeconds(55);
    public static final LocalDateTime FROM_NOW_58 = NOW.plusSeconds(58);
    public static final LocalDateTime FROM_NOW_60 = NOW.plusSeconds(60);
    public static final LocalDateTime FROM_NOW_65 = NOW.plusSeconds(65);
    public static final LocalDateTime FROM_NOW_95 = NOW.plusSeconds(95);
    public static final LocalDateTime FROM_NOW_100 = NOW.plusSeconds(100);
    public static final LocalDateTime FROM_NOW_1000 = NOW.plusSeconds(1000);

    public static final LocalDateTime TODAY =
NOW.withHour(0).withMinute(0).withSecond(0);
    public static final LocalDateTime YESTERDAY = TODAY.minusDays(1);
    public static final LocalDateTime ALMOST_TODAY = TODAY.minusSeconds(1);
    public static final LocalDateTime TODAY_1H = TODAY.plusHours(1);
    public static final LocalDateTime TODAY_2H = TODAY.plusHours(2);
    public static final LocalDateTime TODAY_3H = TODAY.plusHours(3);
    public static final LocalDateTime TODAY_4H = TODAY.plusHours(4);
    public static final LocalDateTime TOMORROW = TODAY.plusDays(1);
    public static final LocalDateTime ALMOST_TOMORROW = TOMORROW.minusSeconds(1);
}
```

EqualsUtil.java

```
package com.netcracker.eductr.tasks.tests.utils;

import com.netcracker.eductr.tasks.tests.model.Task;
import com.netcracker.eductr.tasks.tests.model.TaskList;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.getTargetClass;
import static com.netcracker.eductr.tasks.tests.utils.TaskListUtils.describeTasks;
import static com.netcracker.eductr.tasks.tests.utils.TaskListUtils.tasksToList;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.ABSTRACT_LIST;


public class EqualsUtil {
  private static List<String> left;
  private static List<Task> extra = new ArrayList<>();
  private static List<Task> missing = new ArrayList<>();

  public static boolean areEqual(List<Task> actual, List<Task> expected) {
    left = expected.stream().map(e -> e.getTitle()).collect(Collectors.toList());
```

```
    extra.addAll(actual);
    extra.removeIf(t -> left.contains(t.getTitle()));

    left = actual.stream().map(e -> e.getTitle()).collect(Collectors.toList());
    missing.addAll(expected);
    missing.removeIf(t -> left.contains(t.getTitle()));
    return extra.size() == 0 && missing.size() == 0;
  }

  public static String getMessage() {
    StringBuilder sb = new StringBuilder();
    if (extra.size() != 0) sb.append("Unexpected tasks found: " + describeTasks(extra));
    if (missing.size() != 0) {
      if (sb.length() > 0) sb.append("\n");
      sb.append("Missing tasks found: " + describeTasks(missing));
    }
    return sb.toString();
  }

  public static boolean hasParent(Types.classTypes type) {
    try {
      return getTargetClass(type).getSuperclass().equals(getTargetClass(ABSTRACT_LIST));
    } catch (ClassNotFoundException e) {

    }
    return false;
  }

  public static boolean sameTypes(Types.classTypes clazz, TaskList taskList) {
    try {
      return getTargetClass(clazz).equals(taskList.getInstance().getClass());
    } catch (ClassNotFoundException e) {
      return false;
    }
  }

  public static boolean areEqual(TaskList list1, TaskList list2) {
    return list1.size() == list2.size() && areEqual(tasksToList(list1),
tasksToList(list2));
  }

  public static boolean areEqual(Task t1, Task t2) {
    return t1 != null && t2 != null && t1.getTitle().equals(t2.getTitle()) &&
t1.getTime().equals(t2.getTime());
  }

  public static boolean areEqual(Collection<Task> list1, Collection<Task> list2) {
    return areEqual(new ArrayList<>(list1), new ArrayList<>(list2));
  }
}
```

TaskListUtils.java

```
package com.netcracker.eductr.tasks.tests.utils;

import com.netcracker.eductr.tasks.tests.model.Task;
import com.netcracker.eductr.tasks.tests.model.TaskList;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

public class TaskListUtils {
    public static List<Task> tasksToList(TaskList taskList) {
        return IntStream.range(0, taskList.size()).mapToObj(i ->
taskList.getTask(i)).collect(Collectors.toList());
    }

    public static String describeTasks(List<Task> target) {
        String s = target
                .stream()
                .map(t -> "Task{title:'" + t.getTitle()
                        + "',time:" + t.getTime()
                        + ",startTime:" + t.getStartTime()
                        + ",endTime:" + t.getEndTime()
                        + ",repeatInterval:" + t.getRepeatInterval()
                        + ",active:" + t.isActive()
                        + "}")
                .collect(Collectors.joining(","));
        return "[" + s + "]";
    }

    public static String describeTasks(TaskList taskList) {
        return describeTasks(tasksToList(taskList));
    }
}
```

TaskUtil.java

```
package com.netcracker.eductr.tasks.tests.utils;

import com.netcracker.eductr.tasks.tests.model.Task;

import java.util.Collections;

import static com.netcracker.eductr.tasks.tests.utils.TaskListUtils.describeTasks;

public class TaskUtil {
    public static String describeTask(Task task) {
        return describeTasks(Collections.singletonList(task));
    }
}
```

AsserExt.java

```
package com.netcracker.eductr.tasks.tests.asserts;

import com.netcracker.eductr.tasks.tests.model.Task;
import com.netcracker.eductr.tasks.tests.model.TaskList;
import org.junit.Assert;

import java.util.Arrays;
import java.util.List;

import static com.netcracker.eductr.tasks.tests.utils.EqualsUtil.areEqual;
import static com.netcracker.eductr.tasks.tests.utils.EqualsUtil.getMessage;
import static com.netcracker.eductr.tasks.tests.utils.TaskListUtils.tasksToList;

public class AssertsExt {
    public static void assertEquals(TaskList actual, List<Task> expected) {
        boolean check = areEqual(tasksToList(actual), expected);
        Assert.assertTrue(getMessage(), check);
    }

    public static void assertEquals(TaskList actual, Task... expected) {
        assertEquals(actual, Arrays.asList(expected));
    }

    public static void assertEquals(TaskList actual, TaskList expected) {
        assertEquals(actual, tasksToList(expected));
    }
}
```

T1_TaskTest2.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.Task;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.BaseCreator.isOld;
import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.DatesTimes.*;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.TASK_BASE;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.NEXT_TIME_AFTER;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T1_TaskTest2 {
  @BeforeClass
  public static void init() {
    Assume.assumeTrue(checkClassExistence(TASK_BASE));
    Assume.assumeTrue(checkMethodExistence(NEXT_TIME_AFTER, TASK_BASE));
  }
```

```
  @Test
  public void part1_testNextNonRepeative() {
    if (isOld()) {
      Task task = new Task("some", 10);
      task.setActive(true);
      Assert.assertEquals("nextTimeAfter 0", 10, task.nextTimeAfter(0));
      Assert.assertEquals("nextTimeAfter 9", 10, task.nextTimeAfter(9));
      Assert.assertEquals("nextTimeAfter 10", -1, task.nextTimeAfter(10));
      Assert.assertEquals("nextTimeAfter 100", -1, task.nextTimeAfter(100));
    } else {
      Task task = create("some", FROM_NOW_10);
      task.setActive(true);
      Assert.assertEquals("nextTimeAfter now", FROM_NOW_10, task.nextTimeAfter(NOW));
      Assert.assertEquals("nextTimeAfter 9 from now", FROM_NOW_10,
task.nextTimeAfter(FROM_NOW_9));
      Assert.assertEquals("nextTimeAfter 10 from now", null,
task.nextTimeAfter(FROM_NOW_10));
      Assert.assertEquals("nextTimeAfter 100 from now", null,
task.nextTimeAfter(FROM_NOW_100));
    }
  }
  @Test
  public void part2_testNextRepeative() {
    if (isOld()) {
      Task task = new Task("some", 10, 100, 20);
      task.setActive(true);
      Assert.assertEquals("nextTimeAfter 0", 10, task.nextTimeAfter(0));
      Assert.assertEquals("nextTimeAfter 9", 10, task.nextTimeAfter(9));
      Assert.assertEquals("nextTimeAfter 30", 50, task.nextTimeAfter(30));
      Assert.assertEquals("nextTimeAfter 40", 50, task.nextTimeAfter(40));
      Assert.assertEquals("nextTimeAfter 10", 30, task.nextTimeAfter(10));
      Assert.assertEquals("nextTimeAfter 95", -1, task.nextTimeAfter(95));
      Assert.assertEquals("nextTimeAfter 100", -1, task.nextTimeAfter(100));
    } else {
      Task task = new Task("some", FROM_NOW_10, FROM_NOW_100, 20);
      task.setActive(true);
      Assert.assertEquals("nextTimeAfter now", FROM_NOW_10, task.nextTimeAfter(NOW));
      Assert.assertEquals("nextTimeAfter 9 from now", FROM_NOW_10,
task.nextTimeAfter(FROM_NOW_9));
      Assert.assertEquals("nextTimeAfter 30 from now", FROM_NOW_50,
task.nextTimeAfter(FROM_NOW_30));
      Assert.assertEquals("nextTimeAfter 40 from now", FROM_NOW_50,
task.nextTimeAfter(FROM_NOW_40));
      Assert.assertEquals("nextTimeAfter 10 from now", FROM_NOW_30,
task.nextTimeAfter(FROM_NOW_10));
      Assert.assertEquals("nextTimeAfter 95 from now", null,
task.nextTimeAfter(FROM_NOW_95));
      Assert.assertEquals("nextTimeAfter 100 from now", null,
task.nextTimeAfter(FROM_NOW_100));
    }
  }
  @Test
  public void part3_testNextInactive() {
    if (isOld()) {
      Task task = new Task("some", 10);
```

```
      task.setActive(false);
      Assert.assertEquals("nextTimeAfter 0", -1, task.nextTimeAfter(0));
    } else {
      Task task = new Task("some", FROM_NOW_10);
      task.setActive(false);
      Assert.assertEquals("nextTimeAfter now", null, task.nextTimeAfter(NOW));
    }
  }
}
```

T2_ArrayTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.Task;
import com.netcracker.eductr.tasks.tests.model.TaskList;
import com.netcracker.eductr.tasks.tests.utils.Types;
import org.junit.*;
import org.junit.runners.MethodSorters;

import java.util.Arrays;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import static com.netcracker.eductr.tasks.tests.asserts.AssertsExt.assertEquals;
import static com.netcracker.eductr.tasks.tests.model.BaseCreator.isOld;
import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.DatesTimes.*;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.ARRAY_LIST;
import static com.netcracker.eductr.tasks.tests.utils.TaskListUtils.describeTasks;
import static com.netcracker.eductr.tasks.tests.utils.TaskUtil.describeTask;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.TASK_BASE;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.*;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T2_ArrayTest {

    @BeforeClass
    public static void init() {
        Assume.assumeTrue(checkClassExistence(TASK_BASE));
        Assume.assumeTrue(checkClassExistence(ARRAY_LIST));
        Assume.assumeTrue(checkMethodExistence(SIZE, ARRAY_LIST));
        Assume.assumeTrue(checkMethodExistence(ADD, ARRAY_LIST));
        Assume.assumeTrue(checkMethodExistence(GET_TASK, ARRAY_LIST));
    }

    @Test
    public void part1_testSizeAddGet() {
        TaskList actual = new TaskList();
        Assert.assertEquals("ArrayTaskList.size()", 0, actual.size());
```

```
        List<Task> expected = Arrays.asList(
                create("A"),
                create("B"),
                create("C"));

        expected.forEach(actual::add);
        Assert.assertEquals("ArrayTaskList.size()", 3, actual.size());

        assertEquals(actual, expected);
    }

    @Test
    public void part2_testRemove() {
        TaskList actual = new TaskList();
        Task[] expected;
        if (isOld()) {
            expected = new Task[]{
                    create("A", 42),
                    create("B", 42),
                    create("C", 42),
                    create("D", 42),
                    create("E", 42)};
        } else {
            expected = new Task[]{
                    create("A", FROM_NOW_42),
                    create("B", FROM_NOW_42),
                    create("C", FROM_NOW_42),
                    create("D", FROM_NOW_42),
                    create("E", FROM_NOW_42)};
        }
        Arrays.stream(expected).forEach(actual::add);

        // remove first
        Assert.assertTrue("ArrayTaskList.remove(A) повинно бути істинним",
actual.remove(expected[0]));
        Assert.assertEquals("ArrayTaskList.size()", 4, actual.size());
        assertEquals(actual, expected[1], expected[2], expected[3], expected[4]);

        // remove last
        Assert.assertTrue("ArrayTaskList.remove(E) повинно бути істинним",
actual.remove(expected[4]));
        Assert.assertEquals("ArrayTaskList.size()", 3, actual.size());
        assertEquals(actual, expected[1], expected[2], expected[3]);

        // remove middle
        Assert.assertTrue("ArrayTaskList.remove(C) повинно бути істинним",
actual.remove(expected[2]));
        Assert.assertEquals("ArrayTaskList.size()", 2, actual.size());
        assertEquals(actual, expected[1], expected[3]);

        // remove unexistent
        if (isOld()) Assert.assertFalse("ArrayTaskList.remove(D) не повинно бути
істинним", actual.remove(new Task("F", 42)));
        else Assert.assertFalse("ArrayTaskList.remove(D) не повинно бути істинним",
actual.remove(new Task("F", FROM_NOW_42)));
```

```
        // test multiple remove
        actual.add(expected[0]);
        actual.add(expected[0]);
        Assert.assertEquals("ArrayTaskList.size()", 4, actual.size());
        assertEquals(actual, expected[0], expected[1], expected[3]);
        Assert.assertTrue("ArrayTaskList.remove(A) повинно бути істинним",
actual.remove(expected[0]));
        Assert.assertEquals("ArrayTaskList.size()", 3, actual.size());
        assertEquals(actual, expected[0], expected[1], expected[3]);
        Assert.assertTrue("ArrayTaskList.remove(A) повинно бути істинним",
actual.remove(expected[0]));
        Assert.assertEquals("ArrayTaskList.size()", 2, actual.size());
        assertEquals(actual, expected[1], expected[3]);
    }

    @Test(timeout = 1000)
    public void part3_testIncomingInactive() {
        Assume.assumeTrue(isOld());
        TaskList tasks = new TaskList();

        Task[] ts = {create("A", 0, false), create("B", 1, false), create("C", 2,
false)};

        Arrays.stream(ts).forEach(tasks::add);
        TaskList res;
        if (isOld()) {
            res = tasks.incoming(0, 1000);
            Assert.assertEquals("incoming(0, 1000) має бути пустим, але насправді: " +
describeTasks(res), 0, res.size());
        } else {
            res = tasks.incoming(NOW, FROM_NOW_1000);
            Assert.assertEquals("incoming(now, 1000 from now) має бути пустим, але
насправді: " + describeTasks(res), 0, res.size());
        }

    }

    @Test(timeout = 1000)
    public void part4_testIncoming() {
        Assume.assumeTrue(isOld());
        // range: 50 60

        List<Task> ts = Arrays.asList(
                    create("Simple IN", 55, true),
                    create("Simple OUT", 10, true),
                    create("Inactive OUT", 0, 1000, 1, false),
                    create("Simple bound OUT", 50, true),
                    create("Simple bound IN", 60, true),
                    create("Repeat inside IN", 51, 58, 2, true),
                    create("Repeat outside IN", 0, 100, 5, true),
                    create("Repeat outside OUT", 0, 100, 22, true),
                    create("Repeat left OUT", 0, 40, 1, true),
                    create("Repeat right OUT", 65, 100, 1, true),
                    create("Repeat left intersect IN 1", 0, 55, 13, true),
                    create("Repeat left intersect IN 2", 0, 60, 30, true),
                    create("Repeat left intersect OUT", 0, 55, 22, true),
```

```
                    create("Repeat right intersect IN", 55, 100, 20, true));


        TaskList tasks = new TaskList();
        ts.forEach(tasks::add);
        Set<String> incoming = ts.stream().map(Task::getTitle).filter(t ->
t.contains("IN")).collect(Collectors.toSet());
        tasks = tasks.incoming(50, 60);
        IntStream.range(0, tasks.size()).mapToObj(tasks::getTask).forEach(t -> {
            Assert.assertTrue("incoming(" + (isOld() ? "50, 60" : FROM_NOW_50 + ", " +
FROM_NOW_60) + ") не повинно містити " + describeTask(t),
incoming.contains(t.getTitle()));
            incoming.remove(t.getTitle());
        });
    }
}
```

T3_Exceptions.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.TaskList;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.BaseCreator.isOld;
import static com.netcracker.eductr.tasks.tests.model.Task.createWithException;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.*;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.*;
import static com.netcracker.eductr.tasks.tests.utils.Types.constructorTypes.TITLE_TIME;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.GET_TASK;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T3_Exceptions {
    @BeforeClass
    public static void init() {
        Assume.assumeTrue(checkClassExistence(TASK_BASE));
        Assume.assumeTrue(checkClassExistence(ARRAY_LIST));
        Assume.assumeTrue(checkClassExistence(LINKED_LIST));
        Assume.assumeTrue(checkConstructorExistence(TITLE_TIME, TASK_BASE));
        Assume.assumeTrue(checkMethodExistence(GET_TASK, ARRAY_LIST));
        Assume.assumeTrue(checkMethodExistence(GET_TASK, LINKED_LIST));
    }

    @Test(expected = IllegalArgumentException.class)
    public void part1_checkTaskConstructor() throws Throwable {
        if (isOld()) createWithException("A", -1);
        else createWithException("A", null);
    }

    @Test(expected = IndexOutOfBoundsException.class)
    public void part2_checkGetMethods() throws Throwable {
        Assume.assumeTrue(checkClassExistence(ARRAY_LIST));
        new TaskList().getTaskWithException(-1);
```

```
        Assume.assumeTrue(checkClassExistence(LINKED_LIST));
        new TaskList().getTaskWithException(-1);
    }
}
```

T3_LinkedTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.Task;
import com.netcracker.eductr.tasks.tests.model.TaskList;
import org.junit.*;
import org.junit.runners.MethodSorters;

import java.util.Arrays;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import static com.netcracker.eductr.tasks.tests.asserts.AssertsExt.assertEquals;
import static com.netcracker.eductr.tasks.tests.model.BaseCreator.isOld;
import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.DatesTimes.*;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.LINKED_LIST;
import static com.netcracker.eductr.tasks.tests.utils.TaskListUtils.describeTasks;
import static com.netcracker.eductr.tasks.tests.utils.TaskUtil.describeTask;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.TASK_BASE;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.*;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T3_LinkedTest {
  @BeforeClass
  public static void init() {
        Assume.assumeTrue(checkClassExistence(TASK_BASE));
    Assume.assumeTrue(checkClassExistence(LINKED_LIST));
        Assume.assumeTrue(checkMethodExistence(SIZE, LINKED_LIST));
        Assume.assumeTrue(checkMethodExistence(ADD, LINKED_LIST));
        Assume.assumeTrue(checkMethodExistence(GET_TASK, LINKED_LIST));
  }

    @Test
    public void part1_testSizeAddGet() {
        TaskList actual = new TaskList();
        Assert.assertEquals("LinkedTaskList.size()", 0, actual.size());
        List<Task> expected = Arrays.asList(
                create("A"),
                create("B"),
                create("C"));

      expected.forEach(actual::add);
        Assert.assertEquals("LinkedTaskList.size()", 3, actual.size());
```

```
        assertEquals(actual, expected);
    }

    @Test
    public void part2_testRemove() {
        TaskList actual = new TaskList();
        Task[] expected = {
                create("A"),
                create("B"),
                create("C"),
                create("D"),
                create("E")};
      Arrays.stream(expected).forEach(actual::add);

        // remove first
        Assert.assertTrue("LinkedTaskList.remove(A) повинно бути істинним",
actual.remove(expected[0]));
        Assert.assertEquals("LinkedTaskList.size()", 4, actual.size());
        assertEquals(actual, expected[1], expected[2], expected[3], expected[4]);


        // remove last
        Assert.assertTrue("LinkedTaskList.remove(E) повинно бути істинним",
actual.remove(expected[4]));
        Assert.assertEquals("LinkedTaskList.size()", 3, actual.size());
        assertEquals(actual, expected[1], expected[2], expected[3]);

        // remove middle
        Assert.assertTrue("LinkedTaskList.remove(C) повинно бути істинним",
actual.remove(expected[2]));
        Assert.assertEquals("LinkedTaskList.size()", 2, actual.size());
        assertEquals(actual, expected[1], expected[3]);

        // remove unexistent
        Assert.assertFalse("LinkedTaskList.remove(D) не повинно бути істинним",
actual.remove(create("F")));

        // test multiple remove
        actual.add(expected[0]);
        actual.add(expected[0]);
        Assert.assertEquals("LinkedTaskList.size()", 4, actual.size());
        assertEquals(actual, expected[0], expected[1], expected[3]);
        Assert.assertTrue("LinkedTaskList.remove(A) повинно бути істинним",
actual.remove(expected[0]));
        Assert.assertEquals("LinkedTaskList.size()", 3, actual.size());
        assertEquals(actual, expected[0], expected[1], expected[3]);
        Assert.assertTrue("LinkedTaskList.remove(A) повинно бути істинним",
actual.remove(expected[0]));
        Assert.assertEquals("LinkedTaskList.size()", 2, actual.size());
        assertEquals(actual, expected[1], expected[3]);
    }

    @Test(timeout = 1000)
    public void part3_testIncomingInactive() {
        Assume.assumeTrue(isOld());
```

```
        TaskList tasks = new TaskList();

        Task[] ts = {create("A", 0, false), create("B", 1, false), create("C", 2,
false)};

        Arrays.stream(ts).forEach(tasks::add);
        TaskList res;
        if (isOld()) {
            res = tasks.incoming(0, 1000);
            Assert.assertEquals("incoming(0, 1000) має бути пустим, але насправді: " +
describeTasks(res), 0, res.size());
        } else {
            res = tasks.incoming(NOW, FROM_NOW_1000);
            Assert.assertEquals("incoming(now, 1000 from now) має бути пустим, але
насправді: " + describeTasks(res), 0, res.size());
        }

    }

    @Test(timeout = 1000)
    public void part4_testIncoming() {
        Assume.assumeTrue(isOld());
        // range: 50 60
        List<Task> ts = Arrays.asList(
                create("Simple IN", 55, true),
                create("Simple OUT", 10, true),
                create("Inactive OUT", 0, 1000, 1, false),
                create("Simple bound OUT", 50, true),
                create("Simple bound IN", 60, true),
                create("Repeat inside IN", 51, 58, 2, true),
                create("Repeat outside IN", 0, 100, 5, true),
                create("Repeat outside OUT", 0, 100, 22, true),
                create("Repeat left OUT", 0, 40, 1, true),
                create("Repeat right OUT", 65, 100, 1, true),
                create("Repeat left intersect IN 1", 0, 55, 13, true),
                create("Repeat left intersect IN 2", 0, 60, 30, true),
                create("Repeat left intersect OUT", 0, 55, 22, true),

                create("Repeat right intersect IN", 55, 100, 20, true));

        TaskList tasks = new TaskList();
        ts.forEach(tasks::add);
        Set<String> incoming = ts.stream().map(t -> t.getTitle()).filter(t ->
t.contains("IN")).collect(Collectors.toSet());

        tasks = tasks.incoming(50, 60);
        IntStream.range(0, tasks.size()).mapToObj(tasks::getTask).forEach(t -> {
            Assert.assertTrue("incoming(" + (isOld() ? "50, 60" : FROM_NOW_50 + ", " +
FROM_NOW_60) + ") не повинно містити " + describeTask(t),
incoming.contains(t.getTitle()));
            incoming.remove(t.getTitle());
        });
    }
}
```

T4_AbstractTaskList.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.TaskListFactory;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.BaseCreator.isOld;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.*;
import static com.netcracker.eductr.tasks.tests.utils.EqualsUtil.hasParent;
import static com.netcracker.eductr.tasks.tests.model.ListTypes.count;
import static com.netcracker.eductr.tasks.tests.model.ListTypes.isRightNames;
import static com.netcracker.eductr.tasks.tests.utils.EqualsUtil.sameTypes;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.*;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.*;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T4_AbstractTaskList {
    @BeforeClass
    public static void init() {
        Assume.assumeTrue(checkClassExistence(ARRAY_LIST));
        Assume.assumeTrue(checkClassExistence(LINKED_LIST));
        Assume.assumeTrue(checkClassExistence(ABSTRACT_LIST));
        Assume.assumeTrue(checkClassExistence(LIST_TYPES));
        Assume.assumeTrue(checkClassExistence(TASK_LIST_FACTORY));
    }

    @Test
    public void part1_checkInheritance() {
        Assert.assertTrue("ArrayTaskList повинен наслідуватись від AbstractTaskList",
hasParent(ARRAY_LIST));
        Assert.assertTrue("LinkedTaskList повинен наслідуватись від AbstractTaskList",
hasParent(LINKED_LIST));
    }

    @Test
    public void part2_checkMethodExistence() {
        Assert.assertTrue("AbstractTaskList повинен містити метод size()",
checkDeclaredMethodExistence(SIZE, ABSTRACT_LIST));
        Assert.assertTrue("AbstractTaskList повинен містити метод getTask()",
checkDeclaredMethodExistence(GET_TASK, ABSTRACT_LIST));
        Assert.assertTrue("AbstractTaskList повинен містити метод add()",
checkDeclaredMethodExistence(ADD, ABSTRACT_LIST));
        Assert.assertTrue("AbstractTaskList повинен містити метод remove()",
checkDeclaredMethodExistence(REMOVE, ABSTRACT_LIST));
        if (isOld()) Assert.assertTrue("AbstractTaskList повинен містити метод
incoming()", checkDeclaredMethodExistence(INCOMING, ABSTRACT_LIST));
    }

    @Test
    public void part3_checkTypes() {
        Assert.assertEquals("ListTypes.types повинен мати 2 елемента", 2, count());
        Assert.assertTrue("ListTypes.types повинен мати значення ARRAY і LINKED",
isRightNames());
    }
```

```
    @Test
    public void part4_checkFactory() {
        Assert.assertTrue("TaskListFactory.createTaskList(ARRAY) повинен повертати
об'єкт класу ArrayTaskList", sameTypes(ARRAY_LIST,
TaskListFactory.createTaskList(ARRAY_LIST)));
        Assert.assertTrue("TaskListFactory.createTaskList(LINKED) повинен повертати
об'єкт класу LinkedTaskList", sameTypes(LINKED_LIST,
TaskListFactory.createTaskList(LINKED_LIST)));
    }
}
```

T5_ArrayTaskListCloneTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.TaskList;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.EqualsUtil.areEqual;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.ARRAY_LIST;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.CLONE;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T5_ArrayTaskListCloneTest {
  @BeforeClass
  public static void init() {
    Assume.assumeTrue(checkClassExistence(ARRAY_LIST));
    Assume.assumeTrue(checkMethodExistence(CLONE, ARRAY_LIST));
  }

  private TaskList createList() {
    TaskList tasks = new TaskList();
    tasks.add(create("A"));
    tasks.add(create("B"));
    tasks.add(create("C"));
    return tasks;
  }

  @Test(timeout = 100)
  public void part1_testClone() {
    TaskList original = createList();
    TaskList copy = original.clone();

    Object origRef = original.getInstance();
    Object cloneRef = copy.getInstance();

    Assert.assertTrue("{ x.clone() != x } не виконується", origRef != cloneRef);
```

```
    Assert.assertEquals("{ x.clone().getClass() == x.getClass() } не виконується",
origRef.getClass(), cloneRef.getClass());

    Assert.assertTrue("{ x.clone().equals(x) } не виконується", areEqual(original,
copy));
  }

  @Test(timeout = 100)
  public void part2_testCloneIndependenceRemove() {
    TaskList original = createList();
    TaskList etalon = createList();
    TaskList copy = original.clone();

    copy.remove(original.getTask(0));
    Assert.assertTrue("Після зміни копії оригінал також змінився", areEqual(original,
etalon));
  }

  @Test(timeout = 100)
  public void part3_testCloneIndependenceAdd() {
    TaskList original = createList();
    TaskList etalon = createList();
    TaskList copy = original.clone();

    copy.add(create("D"));
    Assert.assertTrue("Після зміни копії оригінал також змінився", areEqual(original,
etalon));
  }
}
```

T5_ArrayTaskListEqualsTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.TaskList;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.TaskListCreator.*;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.TaskListUtils.describeTasks;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.ARRAY_LIST;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.*;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T5_ArrayTaskListEqualsTest {
  @BeforeClass
  public static void init() {
    Assume.assumeTrue(checkClassExistence(ARRAY_LIST));
    Assume.assumeTrue(checkMethodExistence(HASH_CODE, ARRAY_LIST));
    Assume.assumeTrue(checkMethodExistence(EQUALS, ARRAY_LIST));
  }
```

```
  @Test
  public void part1_testEqualsToItself() {
    TaskList a = new TaskList();
    Assert.assertTrue("a.equals(a), a != null не виконується для " + describeTasks(a),
a.equals(a));
  }

  @Test
  public void part2_testEquals() {
    TaskList a = new TaskList();
    TaskList b = new TaskList();

    Assert.assertTrue("Об'єкти мають бути рівні: " + describeTasks(a) + " та " +
describeTasks(b), a.equals(b));
    Assert.assertTrue("a = b <=> b = a не виконується для " + describeTasks(a) + " та "
+ describeTasks(b), b.equals(a));
  }

  @Test
  public void part3_testEqualsChanged() {
    TaskList a = createA();
    TaskList b = createB();
    modify(a);
    modify(b);

    Assert.assertTrue("Об'єкти мають бути рівні: " + describeTasks(a) + " та " +
describeTasks(b), a.equals(b));
  }

  @Test
  public void part4_testEqualsNull() {
    TaskList a = createA();
    Assert.assertFalse("x.equals(null) == false не виконується", a.equals(null));
  }

  @Test
  public void part5_testEqualsToString() {
    TaskList a = createA();

    Assert.assertFalse("Об'єкти не повинні дорівнювати об'єктам зовсім іншого типу",
a.equals("some string"));
  }

  @Test
  public void part6_testHashCode() {
    TaskList a = createA();
    TaskList b = createB();
    modify(a);
    modify(b);

    Assert.assertTrue("Хеш-коди для " + describeTasks(a) + " (" + a.hashCode() + ") та "
        + describeTasks(b) + " (" + b.hashCode() + ") мають бути рівні", a.hashCode() ==
b.hashCode());
  }
}
```

T5_ArrayTaskListIterator.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.Task;
import com.netcracker.eductr.tasks.tests.model.TaskList;
import org.junit.*;
import org.junit.runners.MethodSorters;

import java.util.*;
import java.util.stream.Collectors;

import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.TaskUtil.describeTask;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.ARRAY_LIST;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.ITERATOR;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T5_ArrayTaskListIteratorTest {
  @BeforeClass
  public static void init() {
    Assume.assumeTrue(checkClassExistence(ARRAY_LIST));
    Assume.assumeTrue(checkMethodExistence(ITERATOR, ARRAY_LIST));
  }

  @Test(timeout = 1000)
  public void part1_testIteration() {
    List<Task> array = Arrays.asList(create("A"), create("B"), create("C"),
create("D"));
    TaskList tasks = new TaskList();
    Set<String> titles = array.stream().map(t ->
t.getTitle()).collect(Collectors.toSet());
    array.forEach(tasks::add);
    for (Iterator<Task> it = tasks.iterator(); it.hasNext();) {
      Task t = new Task(it.next());
      String title = t.getTitle();
      String description = describeTask(t);
      Assert.assertTrue("При ітерації знайдено невідому задачу: " + description,
titles.contains(title));
      titles.remove(title);
    }
    Assert.assertTrue("При ітерації не знайдено наступні задачі: " + titles,
titles.isEmpty());
  }

  @Test(timeout = 1000)
  public void part2_testIteratorRemove() {
    List<Task> array = Arrays.asList(create("A"), create("B"), create("C"),
create("D"));
    TaskList tasks = new TaskList();
    array.forEach(tasks::add);

    // saving order
    List<String> etalon = new ArrayList<>(array.size());
```

```
    for (Iterator<Task> it = tasks.iterator(); it.hasNext();) {
      etalon.add(new Task(it.next()).getTitle());
    }

    Iterator<Task> it = tasks.iterator();
    Iterator<String> etalonIt = etalon.iterator();
    try {
      it.remove();
      Assert.fail("Виклик Iterator.remove без next повинен призводити до помилки");
    }
    catch (IllegalStateException e) {
      // OK
    }

    String actual = new Task(it.next()).getTitle();
    String expected = etalonIt.next();
    Assert.assertEquals("При ітерації задачі повинні зберігати порядок. Задача " +
actual + " знаходиться не на своєму місці", expected, actual);
    it.remove(); etalonIt.remove();
    actual = new Task(it.next()).getTitle();
    expected = etalonIt.next();
    Assert.assertEquals("При ітерації задачі повинні зберігати порядок. Задача " +
actual + " знаходиться не на своєму місці", expected, actual);
    it.next(); etalonIt.next();
    it.remove(); etalonIt.remove();
    it.next(); etalonIt.next();

    it = tasks.iterator();
    etalonIt = etalon.iterator();
    actual = new Task(it.next()).getTitle();
    expected = etalonIt.next();
    Assert.assertEquals("Неочікуваний перший елемент після видалення", expected,
actual);
    actual = new Task(it.next()).getTitle();
    expected = etalonIt.next();
    Assert.assertEquals("Неочікуваний Другий елемент після видалення", expected,
actual);
    Assert.assertTrue("Після видалення у списку має бути два елемента",
etalonIt.hasNext() == it.hasNext());
  }

  @Test(timeout = 1000)
  public void part3_testParallelIteration() {
    TaskList tasks = new TaskList();
    Arrays.asList(create("A"), create("B"), create("C")).forEach(tasks::add);
    Set<String> pairs = new HashSet<>(
        Arrays.asList( "AA", "AB", "AC", "BA", "BB", "BC", "CA", "CB", "CC" ));

    for (Iterator<Task> outer = tasks.iterator(); outer.hasNext();) {
      String out = new Task(outer.next()).getTitle();
      for (Iterator<Task> inner = tasks.iterator(); inner.hasNext();) {
        String in = new Task(inner.next()).getTitle();
        String it = out + in;
        Assert.assertTrue("Паралельна ітерація не повинна призводити до " + it,
pairs.contains(it));
        pairs.remove(it);
```

```
      }
    }

    Assert.assertTrue("Паралельна ітерація повинна призводити до " + pairs,
pairs.isEmpty());
  }
}
```

T5_LinkedTaskListCloneTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.TaskList;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.EqualsUtil.areEqual;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.LINKED_LIST;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.CLONE;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T5_LinkedTaskListCloneTest {
  @BeforeClass
  public static void init() {
    Assume.assumeTrue(checkClassExistence(LINKED_LIST));
    Assume.assumeTrue(checkMethodExistence(CLONE, LINKED_LIST));
  }

  private TaskList createList() {
    TaskList tasks = new TaskList();
    tasks.add(create("A"));
    tasks.add(create("B"));
    tasks.add(create("C"));
    return tasks;
  }

  @Test(timeout = 100)
  public void part1_testClone() {
    TaskList original = createList();
    TaskList copy = original.clone();

    Object origRef = original.getInstance();
    Object cloneRef = copy.getInstance();

    Assert.assertTrue("{ x.clone() != x } не виконується", origRef != cloneRef);

    Assert.assertEquals("{ x.clone().getClass() == x.getClass() } не виконується",
origRef.getClass(), cloneRef.getClass());

    Assert.assertTrue("{ x.clone().equals(x) } не виконується", areEqual(original,
copy));
```

```
  }

  @Test(timeout = 100)
  public void part2_testCloneIndependenceRemove() {
    TaskList original = createList();
    TaskList etalon = createList();
    TaskList copy = original.clone();

    copy.remove(original.getTask(0));
    Assert.assertTrue("Після зміни копії оригінал також змінився", areEqual(original,
etalon));
  }

  @Test(timeout = 100)
  public void part3_testCloneIndependenceAdd() {
    TaskList original = createList();
    TaskList etalon = createList();
    TaskList copy = original.clone();

    copy.add(create("D"));
    Assert.assertTrue("Після зміни копії оригінал також змінився", areEqual(original,
etalon));
  }
}
```

T5_LinkedTaskListEqualsTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.TaskList;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.TaskListCreator.*;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.TaskListUtils.describeTasks;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.LINKED_LIST;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.*;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T5_LinkedTaskListEqualsTest {
  @BeforeClass
  public static void init() {
    Assume.assumeTrue(checkClassExistence(LINKED_LIST));
    Assume.assumeTrue(checkMethodExistence(HASH_CODE, LINKED_LIST));
    Assume.assumeTrue(checkMethodExistence(EQUALS, LINKED_LIST));
  }

  @Test
  public void part1_testEqualsToItself() {
    TaskList a = new TaskList();
    Assert.assertTrue("a.equals(a), a != null не виконується для " + describeTasks(a),
a.equals(a));
```

```
  }

  @Test
  public void part2_testEquals() {
    TaskList a = new TaskList();
    TaskList b = new TaskList();

    Assert.assertTrue("Об'єкти мають бути рівні: " + describeTasks(a) + " та " +
describeTasks(b), a.equals(b));
    Assert.assertTrue("a = b <=> b = a не виконується для " + describeTasks(a) + " та "
+ describeTasks(b), b.equals(a));
  }

  @Test
  public void part3_testEqualsChanged() {
    TaskList a = createA();
    TaskList b = createB();
    modify(a);
    modify(b);

    Assert.assertTrue("Об'єкти мають бути рівні: " + describeTasks(a) + " та " +
describeTasks(b), a.equals(b));
  }

  @Test
  public void part4_testEqualsNull() {
    TaskList a = createA();
    Assert.assertFalse("x.equals(null) == false уе виконується", a.equals(null));
  }

  @Test
  public void part5_testEqualsToString() {
    TaskList a = createA();

    Assert.assertFalse("Об'єкти не повинні дорівнювати об'єктам зовсім іншого типу",
a.equals("some string"));
  }

  @Test
  public void part6_testHashCode() {
    TaskList a = createA();
    TaskList b = createB();
    modify(a);
    modify(b);

    Assert.assertTrue("Хеш-коди для " + describeTasks(a) + " (" + a.hashCode() + ") та "
        + describeTasks(b) + " (" + b.hashCode() + ") мають бути рівні", a.hashCode() ==
b.hashCode());
  }
}
```

T5_LinkedTaskListIteratorTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.Task;
import com.netcracker.eductr.tasks.tests.model.TaskList;
import org.junit.*;
import org.junit.runners.MethodSorters;

import java.util.*;
import java.util.stream.Collectors;

import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.TaskUtil.describeTask;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.LINKED_LIST;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.ITERATOR;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T5_LinkedTaskListIteratorTest {
  @BeforeClass
  public static void init() {
    Assume.assumeTrue(checkClassExistence(LINKED_LIST));
    Assume.assumeTrue(checkMethodExistence(ITERATOR, LINKED_LIST));
  }

  @Test(timeout = 1000)
  public void part1_testIteration() {
    List<Task> array = Arrays.asList(create("A"), create("B"), create("C"),
create("D"));
    TaskList tasks = new TaskList();
    Set<String> titles = array.stream().map(t ->
t.getTitle()).collect(Collectors.toSet());
    array.forEach(tasks::add);
    for (Iterator<Task> it = tasks.iterator(); it.hasNext();) {
      Task t = new Task(it.next());
      String title = t.getTitle();
      String description = describeTask(t);
      Assert.assertTrue("При ітерації знайдено невідому задачу: " + description,
titles.contains(title));
      titles.remove(title);
    }
    Assert.assertTrue("При ітерації не знайдено наступні задачі: " + titles,
titles.isEmpty());
  }

  @Test(timeout = 1000)
  public void part2_testIteratorRemove() {
    List<Task> array = Arrays.asList(create("A"), create("B"), create("C"),
create("D"));
    TaskList tasks = new TaskList();
    array.forEach(tasks::add);

    // saving order
```

```
    List<String> etalon = new ArrayList<>(array.size());
    for (Iterator<Task> it = tasks.iterator(); it.hasNext(); etalon.add(new
Task(it.next()).getTitle()));


    Iterator<Task> it = tasks.iterator();
    Iterator<String> etalonIt = etalon.iterator();
    try {
      it.remove();
      Assert.fail("Виклик Iterator.remove без next повинен призводити до помилки");
    }
    catch (IllegalStateException e) {
      // OK
    }

    String actual = new Task(it.next()).getTitle();
    String expected = etalonIt.next();
    Assert.assertEquals("При ітерації задачі повинні зберігати порядок. Задача " +
actual + " знаходиться не на своєму місці", expected, actual);
    it.remove(); etalonIt.remove();
    actual = new Task(it.next()).getTitle();
    expected = etalonIt.next();
    Assert.assertEquals("При ітерації задачі повинні зберігати порядок. Задача " +
actual + " знаходиться не на своєму місці", expected, actual);
    it.next(); etalonIt.next();
    it.remove(); etalonIt.remove();
    it.next(); etalonIt.next();

    it = tasks.iterator();
    etalonIt = etalon.iterator();
    actual = new Task(it.next()).getTitle();
    expected = etalonIt.next();
    Assert.assertEquals("Неочікуваний перший елемент після видалення", expected,
actual);
    actual = new Task(it.next()).getTitle();
    expected = etalonIt.next();
    Assert.assertEquals("Неочікуваний Другий елемент після видалення", expected,
actual);
    Assert.assertTrue("Після видалення у списку має бути два елемента",
etalonIt.hasNext() == it.hasNext());
  }

  @Test(timeout = 1000)
  public void part3_testParallelIteration() {
    TaskList tasks = new TaskList();
    Arrays.asList(create("A"), create("B"), create("C")).forEach(tasks::add);
    Set<String> pairs = new HashSet<>(
        Arrays.asList( "AA", "AB", "AC", "BA", "BB", "BC", "CA", "CB", "CC" ));

    for (Iterator<Task> outer = tasks.iterator(); outer.hasNext();) {
      String out = new Task(outer.next()).getTitle();
      for (Iterator<Task> inner = tasks.iterator(); inner.hasNext();) {
        String in = new Task(inner.next()).getTitle();
        String it = out + in;
        Assert.assertTrue("Паралельна ітерація не повинна призводити до " + it,
pairs.contains(it));
```

```
        pairs.remove(it);
      }
    }

    Assert.assertTrue("Паралельна ітерація повинна призводити до " + pairs,
pairs.isEmpty());
  }
}
```

T5_TaskCloneTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.Task;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.EqualsUtil.areEqual;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.TASK_BASE;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.CLONE;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T5_TaskCloneTest {
  @BeforeClass
  public static void init() {
    Assume.assumeTrue(checkClassExistence(TASK_BASE));
    Assume.assumeTrue(checkMethodExistence(CLONE, TASK_BASE));
  }

  @Test
  public void part1_testClone() {
    Task original = create("A");
    Task copy = original.clone();

    Object origRef = original.getInstance();
    Object cloneRef = copy.getInstance();

    Assert.assertTrue("{ x.clone() != x } не виконується", origRef != cloneRef);

    Assert.assertEquals("{ x.clone().getClass() == x.getClass() } не виконується",
origRef.getClass(), cloneRef.getClass());

    Assert.assertTrue("{ x.clone().equals(x) } не виконується", areEqual(original,
copy));
  }

  @Test
  public void part2_testCloneIndependence() {
    Task original = create("A");
    Task copy = original.clone();
```

```
    copy.setTitle("Copy");
    Assert.assertNotEquals("Після зміни title копії оригінал також змінився",
original.getTitle(), copy.getTitle());
  }
}
```

T5_TaskEqualsTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.Task;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.TaskCreator.*;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.*;
import static com.netcracker.eductr.tasks.tests.utils.TaskUtil.describeTask;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.TASK_BASE;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.*;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T5_TaskEqualsTest {
  @BeforeClass
  public static void init() {
    Assume.assumeTrue(checkClassExistence(TASK_BASE));
    Assume.assumeTrue(checkDeclaredMethodExistence(HASH_CODE, TASK_BASE));
    Assume.assumeTrue(checkDeclaredMethodExistence(EQUALS, TASK_BASE));
  }

  @Test
  public void part1_testEqualsToItself() {
    Task a = createA();
    Assert.assertTrue("a.equals(a), a != null не виконується для " + describeTask(a),
a.equals(a));
  }

  @Test
  public void part2_testEquals() {
    Task a = createA();
    Task b = createA();

    Assert.assertTrue("Об'єкти мають бути рівні: " + describeTask(a) + " та " +
describeTask(b), a.equals(b));
    Assert.assertTrue("a = b <=> b = a не виконується для " + describeTask(a) + " та " +
describeTask(b), b.equals(a));
  }

  @Test
  public void part3_testEqualsChanged() {
    Task a = createA();
    Task b = createB();
    modify(a);
    modify(b);
```

```
    Assert.assertTrue("Об'єкти мають бути рівні: " + describeTask(a) + " та " +
describeTask(b), a.equals(b));
  }

  @Test
  public void part4_testEqualsNull() {
    Task a = createA();
    Assert.assertFalse("x.equals(null) == false уе виконується", a.equals(null));
  }

  @Test
  public void part5_testEqualsToString() {
    Task a = createA();

    Assert.assertFalse("Об'єкти не повинні дорівнювати об'єктам зовсім іншого типу",
a.equals("some string"));
  }

  @Test
  public void part6_testHashCode() {
    Task a = createA();
    Task b = createB();
    modify(a);
    modify(b);

    Assert.assertTrue("Хеш-коди для " + describeTask(a) + " (" + a.hashCode() + ") та "
        + describeTask(b) + " (" + b.hashCode() + ") мають бути рівні", a.hashCode() ==
b.hashCode());
  }
}
```

T6_StreamAPITest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.TaskList;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.TaskListCreator.createA;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.*;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.*;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.GET_STREAM;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.INCOMING;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T6_StreamAPITest {
  @BeforeClass
  public static void init() {
    Assume.assumeTrue(checkClassExistence(TASK_BASE));
    Assume.assumeTrue(checkClassExistence(ARRAY_LIST));
    Assume.assumeTrue(checkClassExistence(LINKED_LIST));
    Assume.assumeTrue(checkMethodExistence(GET_STREAM, ABSTRACT_LIST));
  }
```

```
  @Test
  public void part1_getStreamTest() throws ClassNotFoundException {
    TaskList.setTargetClass(getTargetClass(ARRAY_LIST));
    Assert.assertEquals("Stream містить неочікувану кількість задач",
createA().getStream().count(), 10);
    TaskList.setTargetClass(getTargetClass(LINKED_LIST));
    Assert.assertEquals("Stream містить неочікувану кількість задач",
createA().getStream().count(), 10);
  }

  @Test
  public void part2_checkMethodDeclaration() {
    Assert.assertFalse("ArrayTaskList не повинен містими метод incoming",
checkDeclaredMethodExistence(INCOMING, ARRAY_LIST));
    Assert.assertFalse("LinkedTaskList не повинен містими метод incoming",
checkDeclaredMethodExistence(INCOMING, LINKED_LIST));
  }
}
```

T7_TasksTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.Task;
import com.netcracker.eductr.tasks.tests.model.Tasks;
import org.junit.*;
import org.junit.runners.MethodSorters;

import java.time.LocalDateTime;
import java.util.*;
import java.util.stream.Collectors;

import static com.netcracker.eductr.tasks.tests.model.BaseCreator.isOld;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.DatesTimes.*;
import static com.netcracker.eductr.tasks.tests.utils.EqualsUtil.areEqual;
import static com.netcracker.eductr.tasks.tests.utils.EqualsUtil.getMessage;
import static com.netcracker.eductr.tasks.tests.utils.TaskUtil.describeTask;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.TASKS;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.TASK_BASE;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.INCOMING;
import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T7_TasksTest {
    @BeforeClass
    public static void init() {
        Assume.assumeTrue(checkClassExistence(TASK_BASE));
        Assume.assumeTrue(checkClassExistence(TASKS));
        Assume.assumeTrue(checkMethodExistence(INCOMING, TASKS));
        Assume.assumeFalse(isOld());
    }
```

```
    @Test
    public void part1_testIncomingInactive() {
        List<Task> input = Arrays.asList(create("A", NOW, false), create("B",
FROM_NOW_1, false), create("C", FROM_NOW_3, false));
        Iterable<?> res = Tasks.incoming(input, NOW, FROM_NOW_1000);
        Assert.assertFalse("incoming(" + input + ", " + NOW + ", " + FROM_NOW_1000 + ")
має бути пустим, але насправді: " + res, res.iterator().hasNext());
    }

    @Test
    public void part2_testIncoming() {
        // range: 50 60
        List<Task> ts = Arrays.asList(
                create("Simple IN", FROM_NOW_55, true),
                create("Simple OUT", FROM_NOW_10, true),
                create("Inactive OUT", NOW, FROM_NOW_1000, 1, false),
                create("Simple bound OUT", FROM_NOW_50, true),
                create("Simple bound IN", FROM_NOW_60, true),
                create("Repeat inside IN", FROM_NOW_51, FROM_NOW_58, 2, true),
                create("Repeat outside IN", NOW, FROM_NOW_100, 5, true),
                create("Repeat outside OUT", NOW, FROM_NOW_100, 22, true),
                create("Repeat left OUT", NOW, FROM_NOW_40, 1, true),
                create("Repeat right OUT", FROM_NOW_65, FROM_NOW_100, 1, true),
                create("Repeat left intersect IN 1", NOW, FROM_NOW_55, 13, true),
                create("Repeat left intersect IN 2", NOW, FROM_NOW_60, 30, true),
                create("Repeat left intersect OUT", NOW, FROM_NOW_55, 22, true),
                create("Repeat right intersect IN", FROM_NOW_55, FROM_NOW_100, 20, true)
        );
        List<String> incoming = new ArrayList<>();
        ts.stream().map(Task::getTitle).filter(t ->
t.contains("IN")).forEach(incoming::add);

        Iterable<Task> res = Tasks.incoming(ts, FROM_NOW_50, FROM_NOW_60);
        String call = "incoming(" +
ts.stream().map(Task::getTitle).collect(Collectors.toList()) + ", " + FROM_NOW_50 + ", "
+ FROM_NOW_60 + ")";

        for (Object o : res) {
            Task t = new Task(o);
            Assert.assertTrue(call + " не повинно містити " + describeTask(t),
incoming.contains(t.getTitle()));
            incoming.remove(t.getTitle());
        }
        Assert.assertTrue(call + " повинно містити " + incoming, incoming.isEmpty());
    }

    @Test
    public void part3_testTimeline() {
        Task daily = create("Daily", YESTERDAY, TOMORROW, 3600*24);
        Task hourly = create("Hourly", TODAY, TOMORROW, 3600);
        Task every3h = create("Every 3 hours", TODAY_1H, TOMORROW, 3*3600);
        daily.setActive(true);
        hourly.setActive(true);
        every3h.setActive(true);
```

```
        SortedMap<LocalDateTime, Set<Task>> timeline = new TreeMap<>();
        timeline.put(TODAY, new HashSet<>(Arrays.asList(daily, hourly)));
        timeline.put(TODAY_1H, new HashSet<>(Arrays.asList(hourly, every3h)));
        timeline.put(TODAY_2H, new HashSet<>(Collections.singletonList(hourly)));
        timeline.put(TODAY_3H, new HashSet<>(Collections.singletonList(hourly)));
        timeline.put(TODAY_4H, new HashSet<>(Arrays.asList(hourly, every3h)));

        SortedMap<LocalDateTime, Set<Task>> result = Tasks.calendar(new
HashSet<>(Arrays.asList(daily, hourly, every3h)), ALMOST_TODAY, TODAY_4H);
        Set<LocalDateTime> res = new HashSet<>(result.keySet());
        res.removeAll(timeline.keySet());

        Assert.assertEquals("Неочікувані дати у календарі: " + res, 0, res.size());
        for (LocalDateTime date : timeline.keySet()) {
            List<Task> resList = new ArrayList<>();
            for (Object o : result.get(date)) resList.add(new Task(o));
            if (!areEqual(timeline.get(date), resList))
                Assert.fail("Для " + date + ": " + getMessage());
        }
    }
}
```

T7_TaskTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.Task;
import org.junit.*;
import org.junit.runners.MethodSorters;

import static com.netcracker.eductr.tasks.tests.model.BaseCreator.isOld;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.DatesTimes.*;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.ABSTRACT_LIST;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.TASK_BASE;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.NEXT_TIME_AFTER;
import static com.netcracker.eductr.tasks.tests.model.TaskCreator.create;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T7_TaskTest {
    @BeforeClass
    public static void init() {
        Assume.assumeTrue(checkClassExistence(TASK_BASE));
        Assume.assumeTrue(checkMethodExistence(NEXT_TIME_AFTER, TASK_BASE));
        Assume.assumeTrue(checkClassExistence(ABSTRACT_LIST));
        Assume.assumeFalse(isOld());
    }

    @Test
    public void part1_testNextNonRepeative() {
        Task task = create("some", TODAY);
        task.setActive(true);
```

```
        Assert.assertEquals("nextTimeAfter(YESTERDAY) повернув неочікаву дату", TODAY,
task.nextTimeAfter(YESTERDAY));
        Assert.assertEquals("nextTimeAfter(ALMOST_TODAY) повернув неочікаву дату",
TODAY, task.nextTimeAfter(ALMOST_TODAY));
        Assert.assertNull("nextTimeAfter(TODAY) повернув неочікаву дату",
task.nextTimeAfter(TODAY));
        Assert.assertNull("nextTimeAfter(TOMORROW) повернув неочікаву дату",
task.nextTimeAfter(TOMORROW));
    }

    @Test
    public void part2_testNextRepeative() {
        Task task = create("some", TODAY, TOMORROW, 3600);
        task.setActive(true);
        Assert.assertEquals("nextTimeAfter(YESTERDAY) повернув неочікаву дату", TODAY,
task.nextTimeAfter(YESTERDAY));
        Assert.assertEquals("nextTimeAfter(ALMOST_TODAY) повернув неочікаву дату",
TODAY, task.nextTimeAfter(ALMOST_TODAY));
        Assert.assertEquals("nextTimeAfter(TODAY) повернув неочікаву дату", TODAY_1H,
task.nextTimeAfter(TODAY));
        Assert.assertEquals("nextTimeAfter(TODAY_1H) повернув неочікаву дату", TODAY_2H,
task.nextTimeAfter(TODAY_1H));
        Assert.assertEquals("nextTimeAfter(TODAY_2H) повернув неочікаву дату", TODAY_3H,
task.nextTimeAfter(TODAY_2H));
        Assert.assertEquals("nextTimeAfter(ALMOST_TOMORROW) повернув неочікаву дату",
TOMORROW, task.nextTimeAfter(ALMOST_TOMORROW));
        Assert.assertNull("nextTimeAfter(TOMORROW) повернув неочікаву дату",
task.nextTimeAfter(TOMORROW));
    }

    @Test
    public void part3_testNextInactive() {
        Task task = create("some", TODAY);
        task.setActive(false);
        Assert.assertNull("nextTimeAfter(YESTERDAY) повернув неочікаву дату",
task.nextTimeAfter(YESTERDAY));
    }
}
```

T8_TaskIOTest.java

```
package com.netcracker.eductr.tasks.tests;

import com.netcracker.eductr.tasks.tests.model.TaskIO;
import com.netcracker.eductr.tasks.tests.model.TaskList;
import org.junit.Assume;
import org.junit.BeforeClass;
import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;

import java.io.*;

import static com.netcracker.eductr.tasks.tests.asserts.AssertsExt.assertEquals;
```

```
import static com.netcracker.eductr.tasks.tests.model.BaseCreator.isOld;
import static com.netcracker.eductr.tasks.tests.model.TaskListCreator.createList;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkClassExistence;
import static com.netcracker.eductr.tasks.tests.utils.ClassFinder.checkMethodExistence;
import static com.netcracker.eductr.tasks.tests.utils.Types.classTypes.*;
import static com.netcracker.eductr.tasks.tests.utils.Types.methodTypes.*;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class T8_TaskIOTest {
  @BeforeClass
  public static void init() {
    Assume.assumeTrue(checkClassExistence(TASK_BASE));
    Assume.assumeTrue(checkClassExistence(TASK_IO));
    Assume.assumeTrue(checkClassExistence(ARRAY_LIST));
    Assume.assumeFalse(isOld());
  }

  @Test(timeout = 1000)
  public void part1_testBinary() throws IOException {
    Assume.assumeTrue(checkMethodExistence(WRITE_BINARY, TASK_IO));
    Assume.assumeTrue(checkMethodExistence(READ_BINARY, TASK_IO));
    TaskList expected = createList();

    PipedInputStream in = new PipedInputStream();
    PipedOutputStream out = new PipedOutputStream(in);

    TaskIO.write(expected, out);
    TaskList result = new TaskList();
    TaskIO.read(result, in);

    assertEquals(result, expected);
  }

  @Test(timeout = 1000)
  public void part2_testText() throws IOException {
    Assume.assumeTrue(checkMethodExistence(WRITE, TASK_IO));
    Assume.assumeTrue(checkMethodExistence(READ, TASK_IO));

    TaskList expected = createList();
    TaskList actual = new TaskList();

    TaskIO.write(expected, new FileWriter("test.json"));
    TaskIO.read(actual, new FileReader("test.json"));

    assertEquals(actual, expected);
  }
}
```