Практичні роботи з курсу «Програмування мовою Java»

Зміст

Практична робота 1
Структура проекту

Основні відомості про проект

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

Об'єкти «Задача»

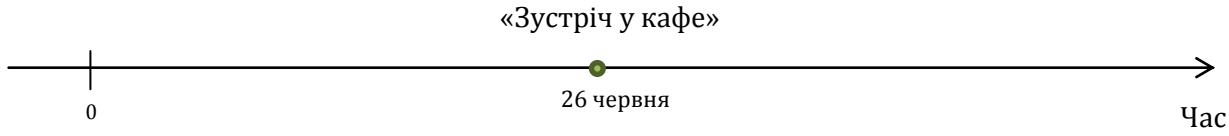
Основними об'єктами, з якими буде працювати додаток, є задачі.

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

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

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

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



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

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

Завдання. Частина 1

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

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

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

Відомості щодо виконання робіт

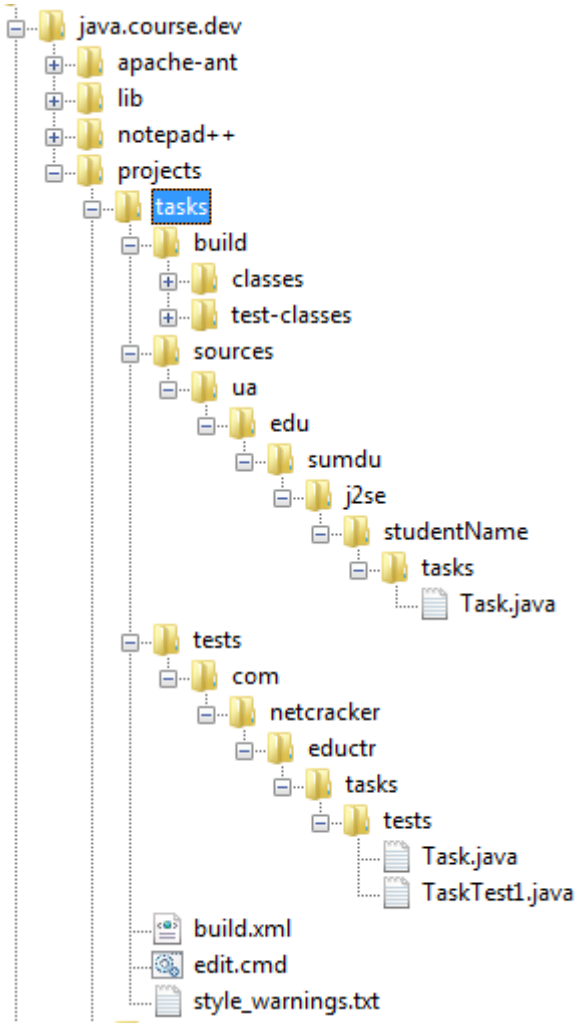
Для виконання роботи вам знадобиться JDK – Java Development Kit для Java SE версії 5 або вище. Завантажити і встановити його можна безкоштовно за адресою http://www.oracle.com/technetwork/java/javase/downloads/index.html.

При виконанні **перших трьох** практичних робіт необхідно використовувати текстовий редактор з підсвічуванням синтаксису, наприклад Notepad++.

Використовувати **середовище розробки** (IDE, наприклад IntelliJ IDEA, Eclipse, NetBeans) дозволяється, починаючи з третьої практичної роботи. В якості IDE в навчальному закладі буде використовуватися IntelliJ IDEA 14.1.4.

Для полегшення роботи пропонується використовувати Notepad++, що настроєний спеціально для Java курсу – *java.course.dev* середовище.

Структура каталогів у каталозі проекту:



- *java.course.dev/projects* – каталог із проектами
- *tasks* – базовий каталог для проекту Task Manager.
- *tasks/sources* – базовий каталог для текстів Java класів.
- *tasks/tests* – базовий каталог для текстів тестів, що перевіряють функціонал основних класів.
- *tasks/build* – базовий каталог для відкомпільованих класів
- *tasks/edit.cmd* – команда для запуску Notepad++

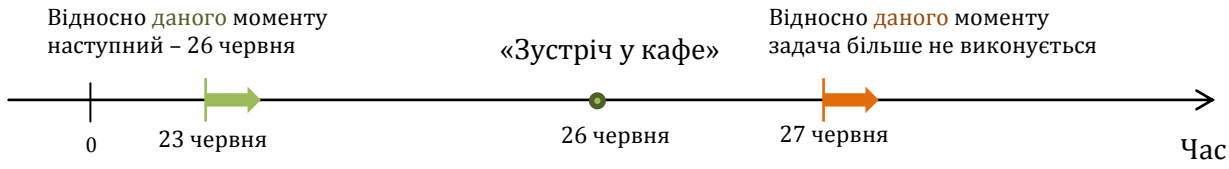
Зверніть увагу на те, що всередині каталогу для текстів java класів *sources* повинна міститися структура каталогів, що відповідає структурі пакетів ваших класів.

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

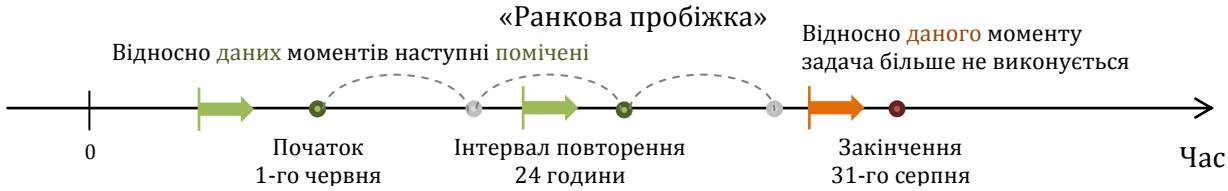
Завдання. Частина 2

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

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



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



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

Практична робота 2
Масиви та посилання

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

Завдання. Частина 1

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

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

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

Завдання. Частина 2

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

Наприклад, нехай у нашому списку знаходяться наступні задачі:

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

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

Практична робота 3
Спадкування, виключні ситуації

Завдання 1. Виключні ситуації

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

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

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

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

Завдання 2. Спадкування

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 і 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`.

1. Оскільки обидва класи 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 у абстрактному класі `TaskList` (при цьому тип об'єкту, що ним повертається також зміниться на `TaskList`), таким чином зменшивши дублювання коду.

Таким чином ми отримаємо:

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

Практична робота 4
Ітератори, сервісні методи

Завдання 1. Ітератори

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

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

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

Завдання 2. Сервісні методи

Імплементувати сервісні методи із класу `Object`:

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

Практична робота 5
Сервісні класи та колекції

Завдання 1. Перехід до використання класу Date

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

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

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

Завдання 2. Виділення логіки роботи з календарем

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

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

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

Наприклад, нехай у нашому списку знаходяться наступні задачі:

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

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

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

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

Практична робота 6

Ввід-вивід та серіалізація

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

Завдання 1. Серіалізація

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

Завдання 2. Бінарний ввід-вивід

Необхідно створити клас `TaskIO` із статичними методами:

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

Формат представлення списку задач у двійковому вигляді:

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, описаному нижче.
- `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 виглядає наприклад наступним чином:

```
"Task title" at [2014-06-28 18:00:13.000];
"Very ""Good"" title" at [2013-05-10 20:31:20.001] inactive;
"Other task" from [2010-06-01 08:00:00.000] to [2010-09-01 00:00:00.000] every [1 day].
```

1. Кожна задача розміщується на окремому рядку.
2. Назва задачі міститься у подвійних лапках, якщо у назві зустрічаються подвійні лапки вони подвоюються. Вважається, що назва задачі завжди складається з одного рядка.
3. Дати форматуються як `[РІК-МІСЯЦЬ-ДЕНЬ ГОДИНА:ХВИЛИНА:СЕКУНДА.МІЛІСЕКУНДА]`.
4. Інтервал повторення форматується як ціла кількість днів, годин, хвилин і секунд використовуючи слова `day(s)`, `hour(s)`, `minute(s)`, `second(s)` (`s` додається якщо число більше 1) через пробіл. Наприклад 10000 секунд буде виглядати як `[2 hours 46 minutes 40 seconds]`.
5. Після кожної задачі іде точка із комою, після останньої – крапка.

Практична робота 7
Робота з консольними утилітами збірки і тестування

Загальні відомості

У цій практичній роботі використовуються консольні команди, що є частиною JDK і містяться у її директорії `bin`. Для використання цих команд без вказування повного шляху необхідно шлях до директорії `bin` у JDK додати до змінної середовища `PATH` у Windows (*Панель керування > Система > Розширені параметри > Змінні середовища*).

Консольні команди можуть виконуватись у консолі (можна запускати командою `cmd` у меню *Пуск > Виконати команду*) або через FAR Manager. Також декілька консольних команд можна записувати у текстовий файл з розширенням .bat для виконання разом.

У .bat файлах можна розбивати команду на декілька рядків використовуючи символ `^` в кінці рядка, що має продовжуватись на наступному. Крім того рекомендується викликати в кінці файлу команду `pause` для того, щоби вікно с результатами виконання команд не зникало після їх виконання.

Консольні команди часто приймають різні аргументи, що записуються через пробіл, якщо аргументи містять пробіли, то їх заключають у подвійні лапки. При посиланні на файли і директорії рекомендується вказувати відносні шляхи (відносно поточної директорії).

Завдання 1. Компіляція

Для компіляції java-кодів використовується утиліта `javac` із JDK, її основні аргументи можна подивитися якщо викликати її із аргументом `-help` (наведено основні параметри):

```
>javac -help
Usage: javac <options> <source files>
where possible options include:
  -classpath <path>          Шлях до існуючих відкомпільованих класів
  -sourcepath <path>         Шлях до вихідних кодів
  -d <directory>             Директорія для створення відкомпільованих класів
```

Тут і далі використовуються аргументи (ключі) із параметрами – після такого аргументу має йти аргумент, що вказує значення цього параметру. Так після аргументу `-d` повинна йти директорія для відкомпільованих класів.

У `classpath` вказуються директорії та .jar архіви із існуючими відкомпільованими класами, що використовуються кодом, що ми компілюємо. Декілька елементів відділяються `;` у windows і `:` у linux.

У `sourcepath` вказується коренева директорія, де знаходяться вихідні коди, там компілятор може самостійно шукати файли для компіляції, якщо їх повні імена відповідають структурі каталогів: наприклад клас `some.test.Class` має знаходитися у файлі `${sourcepath}/some/test/Class.java`.

Необхідно створити файл `compile.bat` у директорії проекту `tasks` для компіляції ваших існуючих класів у директорію `build/classes`. Ваші класи не містять зовнішніх залежностей, тому використовувати ключ `classpath` не потрібно.

Завдання 2. Компіляція тестів

Необхідно створити файл compile-tests.bat для компіляції вихідних файлів тестів у папку `build/test-classes`. Файли тестів залежать від класів, що знаходяться у бібліотеці `java.course.dev/lib/tester.jar`, шлях до неї необхідно вказати відносний - `../../lib/tester.jar`.

Завдання 3. Збірка jar бібліотеки

В результаті компіляції ви отримали директорію із .class файлами, що лежать у директоріях відповідно до пакетів. Для розповсюдження і зручного використання їх необхідно запакувати у jar бібліотеку (Java ARchive), що є спеціальним типом архіву. Для цього використовується утиліта `jar` із JDK, її основні параметри:

```
>jar
Usage: jar {ctxui}[vfm0Me] [jar-file] [manifest-file] [entry-point] [-C dir] files
Options:
    -c  create new archive
    -t  list table of contents for archive
    -x  extract named (or all) files from archive
    -u  update existing archive
    -v  generate verbose output on standard output
    -f  specify archive file name
    -m  include manifest information from specified manifest file
    -e  specify application entry point for stand-alone application
        bundled into an executable jar file
    -0  store only; use no ZIP compression
    -M  do not create a manifest file for the entries
    -i  generate index information for the specified jar files
    -C  change to the specified directory and include the following file
```

Вам необхідно створити файл `build.bat`, що збирає відкомпільовані класи із `build/classes` у `build/tasks.jar`, а із `build/test-classes` у `build/tasks-tests.jar`. При цьому в архівах шляхи повинні починатися відразу із директорій, що відповідають пакетам, а не з `classes` чи `test-classes` (перевірити це можна викликавши `jar tf file.jar` (**t**able of contents of **f**ile). Для цього необхідно використати ключ `-C dir` та вказати файл для архівації `.` – що означає поточну директорію.

Завдання 4. Запуск тестів

Утиліта для запуску тестів знаходиться у бібліотеці `tester.jar` і є java класом `com.netcracker.eductr.tester.runner.TestRunner`. Для запуску java класів використовується утиліта JDK `java`, її основні аргументи:

```
>java
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)

where options include:
    -classpath <class search path of directories and zip/jar files>
                  A ; separated list of directories, JAR archives,
                  and ZIP archives to search for class files.
    -D<name>=<value>
    -ea[:<packagename>...|:<classname>]
```

Клас приймає у якості аргументів sourcepath та імена класів тестів для запуску, окрім того необхідно включити опцію `-ea` та ключ `-Dfile.encoding=cp866`.

Створіть файл `test.bat`, що буде запускати всі тести, ім'я яких починається із `T` використовуючи jar архіви із завдання 3 у якості `classpath` (як доповнення до самого `tester.jar`).

Завдання 5. Створення jar додатка

Java дозволяє створювати jar архіви, що можна запускати як звичайні додатки. Для цього у них додається спеціальний текстовий файл – маніфест.

Створіть клас `PrintMonth` з наступним кодом:

```
package ua.sumdu.j2se.studentName.tasks;

import java.util.*;
import java.text.*;
import static java.util.Calendar.*;
import static java.lang.Integer.parseInt;
import static java.lang.String.format;

public class PrintMonth {
    public static void main(String[] args) {
        Calendar c = Calendar.getInstance();
        c.set(DAY_OF_MONTH, 1);
        c.getTime();
        if (args.length == 1) {
            c.set(MONTH, parseInt(args[0]) - 1);
        }
        int month = c.get(MONTH);
        System.out.println(new SimpleDateFormat("MMMM yyyy:").format(c.getTime()));

        c.set(DAY_OF_WEEK, c.getFirstDayOfWeek());

        c.add(DATE, -7);
        DateFormat wd = new SimpleDateFormat("EE");
        for (int i = 0; i < 7; i++) {
            System.out.print(" ");
            System.out.print(wd.format(c.getTime()));
            c.add(DATE, 1);
        }
        System.out.println();

        do {
            System.out.print(c.get(MONTH) != month ?
                "   " :
                format("%3d", c.get(DAY_OF_MONTH)));
            c.add(DATE, 1);
            if (c.get(DAY_OF_WEEK) == c.getFirstDayOfWeek())
                System.out.println();
        }
        while (c.get(MONTH) == month || c.get(DAY_OF_WEEK) != c.getFirstDayOfWeek());
    }
}
```

Створіть текстовий файл `manifest.mf` у директорії проекту із текстом:

```
Manifest-Version: 1.0
Created-By: Student Name
Main-Class: ua.sumdu.j2se.studentName.tasks.PrintMonth

```

Зверніть увагу на пустий рядок в кінці файлу.

Додайте до файлу `build.bat` необхідні аргументи команди `jar` для використання вказаного маніфесту.

Запустіть ваш додаток використовуючи команду `java -Dfile.encoding=cp866 -jar build\tasks.jar`.

Лабораторна робота 1
Повноцінний додаток «Task Manager»

Загальні відомості

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

Основні вимоги

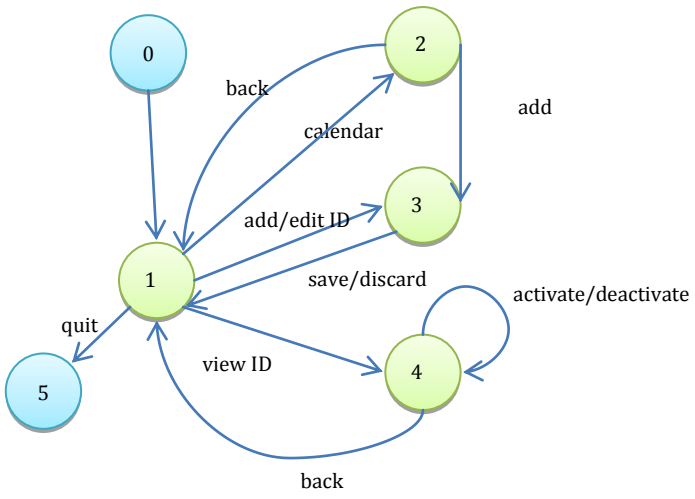
1. Додаток може мати на вибір студента текстовий (консольний) або графічний (GUI) інтерфейс користувача.
2. Додаток повинен зберігати дані між запусками у файловій системі.
3. Додаток повинен надавати користувачу наступні основні функції:
    a. створювати нові задачі;
    b. змінювати параметри існуючих задач;
    c. видаляти задачі;
    d. переглядати інформацію про існуючі задачі;
    e. переглядати календар запланованих подій на деякий проміжок часу.
4. Додаток повинен сповіщати користувача в момент, коли деяка задача має бути виконана.
5. Додаток повинен бути стійкий до помилок: виводити зрозумілі користувачу повідомлення, не втрачати працездатності, контролювати користувацький ввід.
6. Додаток має використовувати бібліотеку Log4j для логування.

Дизайн додатку

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

Communication Diagram

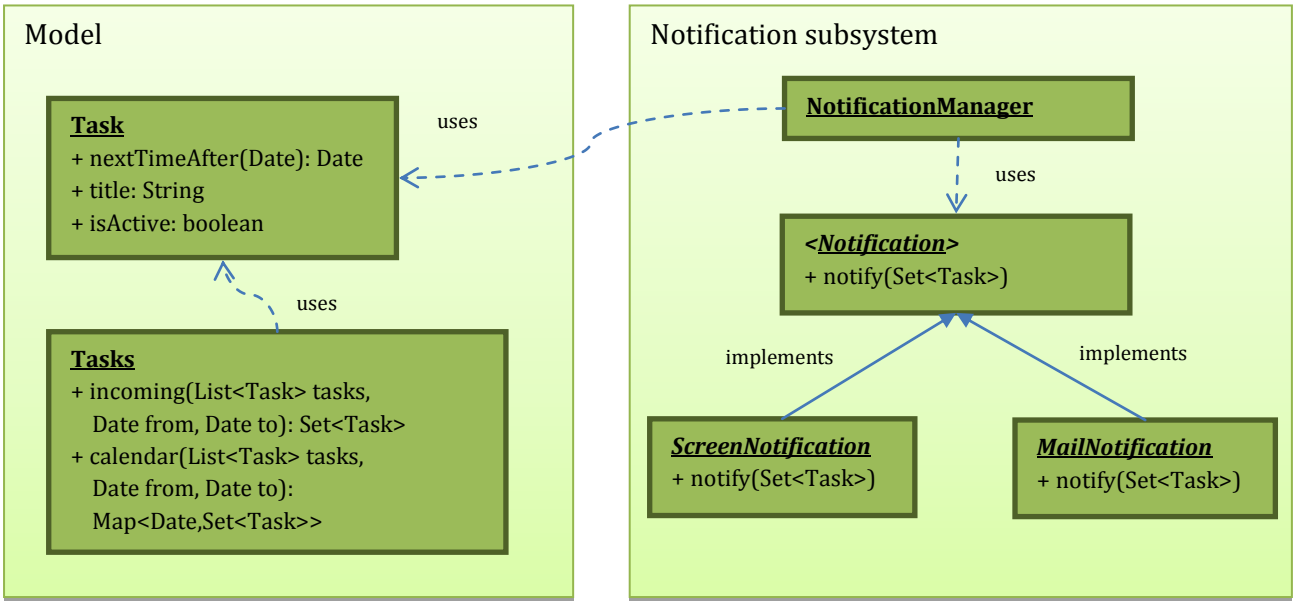
Ця діаграма повинна показувати які дії може виконувати користувач із системою і до чого це призведе, наприклад:



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

CLASS DIAGRAM

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



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