**Библиотека (API) для работы с одномерными и двумерными массивами.**

**Цели задания:**

1. Научиться писать свои собственные методы
2. Научиться писать свои собственные классы
3. Научиться создавать пакеты и импортировать классы из них
4. Познакомиться с JAR-файлами и их структурой
5. Разобраться с понятием **ClassPath** и способами его конфигурации
6. Освоить утилиту **JAR**
7. Научиться создавать командные файлы для компиляции и сборки приложений.

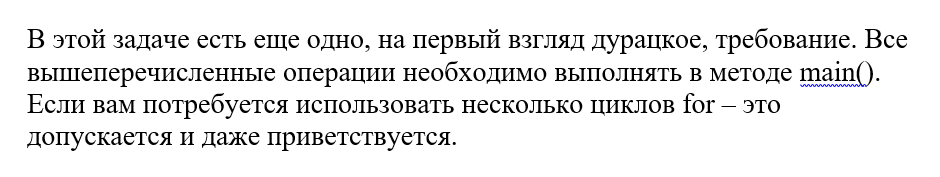
**Описание:**

Друзья, в этой лабораторной работе вам не придется программировать никаких новых алгоритмов, так как текущее задание полностью строится на двух предыдущих (и конечно уже выполненных вами ;)) заданиях:

1. Task-4-2: Одномерные массивы
2. Task-4-3: Двумерные массивы

Возникает резонный вопрос – «Зачем дублировать задачи, если они не дадут нам никаких новых знаний?» Однако это не так. Текущая задача даст вам гораздо больше практических знаний и навыков, чем дали две предыдущие. Потому что те элементы программы и приемы, которые вы изучите в рамках текущей работы в дальнейшем вы будете использовать каждый день, а вот транспонировать матрицу, скорее всего, в следующий раз вы будете только тогда, когда ваши дети поступят в институт ☺ (Конечно, этот сарказм не относится к тем из вас, кто найдет себя в научных проектах или GameDev-е. Ребята, вам честь и хвала, и никакого сарказма).

Также в начале стоит отметить, что текущая работа призвана исправить вот это, **действительно дурацкое, и в корне неправильное** требование предыдущих работ:



В работах 4-2 и 4-3 это требование было введено для того, чтобы у вас появился наглядный пример того, **как делать нельзя**! Действительно – программы не пишутся в одном методе! Вместо этого мы:

* Пишем множество методов, каждый из которых решает свою задачу
* Группируем похожие методы в классы
* Классы организовываем в пакеты

В результате, из всего вышеперечисленного мы можем накопить библиотеки полезных алгоритмов, собрать их в отдельный JAR-файл (или даже файл***ы***) и повторно использовать в любых проектах, которые будем разрабатывать в будущем. Естественно, наша будущая разработка станет быстрее, так как мы будем сосредоточены только на новых требованиях бизнес-логии, а не на переписывании одного и того же алгоритма по десятому разу! Вы не поверите, но примерно так и рождаются в нашем безумном Java-мире новые API и фреймворки ☺

Ок, хватит лить воду, давайте переходить к сути задачи.

**Первое,** что вам нужно сделать – это написать два класса:

1. **ArrayUtils** – класс, который будет содержать все методы для работы с одномерными массивами (см. Task-4-2)
2. **MatrixUtils** – класс, который будет содержать все методы для работы с двумерными массивами (см. Task-4-3)

Оба эти класса необходимо уложить в пакет:

**<ваш\_корневой\_пакет >.arrays.utils**

**Второе,** что вам нужно сделать – это написать командный файл для компиляции и сборки вашей библиотеки:

* **arrays-api-build.bat** – под Windows
* **или arrays-api-build.sh** – под Linux/Mac

Этот командный файл после запуска должен выполнять следующий алгоритм:

1. Удалять предыдущую сборку:
   1. Папку **build**, содержащую ранее скомпилированные классы
   2. И папку **dist**, содержащую ранее собранный JAR-файл библиотеки
2. Создавать пустые папки **build** и **dist**.
3. Компилировать все \*.java файлы в папку **build**.
4. На основании скомпилированных классов из папки build создавать JAR-архив библиотеки в папке **dist**. Архив может называться, например, **arrays-api.jar**

**Внимание:** Начало и окончание каждого этапа сборки должно отражаться в консоли выводом соответствующего сообщения (сообщения придумывайте сами, но чтоб без мата! ☺)

**И третье**, что вам нужно сделать – создать **отдельное приложение** для тестирования вашей библиотеки. Это приложение может называться **arrays-test** и содержать единственный класс с методом main, в котором вы и вызовете все методы из вашей библиотеки.

Куда интереснее здесь то, что это приложение тоже должно собираться в отдельный JAR-файл. Причем это должен быть **Executable JAR**. Для сборки так же необходимо написать отдельный командный файл. Более того, этот командный файл при запуске должен получать от пользователя 2 дополнительных параметра командной строки:

1. Абсолютный путь к корневому каталогу проекта с вашим API (**arrays-api**)
2. Флаг (true/false) того, хотите вы пересобрать API перед сборкой приложения или нет
   1. **Если вы укажете false**, то текущий файл сборки должен просто скопировать уже готовый JAR из папки **dist** вашего API-проекта в папку **lib** текущего приложения.
   2. **Если же вы укажете true**, то текущий файл сборки должен сначала вызвать новую сборку вашего API, и только потом скопировать собранный JAR из папки **dist** вашего API-проекта в папку **lib** текущего приложения.

В остальном командный файл для приложения должен повторить алгоритм командного файла для API. Потому что, как мы помним – алгоритм компиляции и сборки различных приложений во многом – один и тот же. Естественно, если речь идет о простых (учебных) приложениях.

Не забывайте только, что файл MANIFEST.MF для приложения должен содержать два дополнительных параметра:

1. Main-Class – чтобы сделать Executable JAR.
2. И Class-Path – чтобы указать относительный путь к вашей библиотеке.

Вот и все, друзья. Как видите, работы выше крыши, хоть программировать и не пришлось ☺ Тем более, еще раз скажем, что это работа именно с практической направленностью. На любом вашем реальном проекте будет процесс компиляции и сборки со своими уникальными инструментами, алгоритмами, этапами и так далее. Именно со сборки (а не с программирования, как вам могло казаться ранее) вы начнете процесс знакомства с вашим проектом. И именно этот процесс (а не программирование, как вам могло казаться ранее) покажется вам самым большим геморроем, с которым вы когда-либо сталкивались ☺

Но, все равно, избежать его не удастся. Успешная сборка и запуск проекта – будет вашей первой задачей в новой компании. И чем быстрее вы ее решите, тем быстрее начнете свой путь от Junior Developer-а, который испуганно подпрыгивает на своем стуле от звука работающей кофе-машины до архитектора, который может поймать любого в коридоре и небрежно бросить в его сторону – «Эй, человек… Кофе мне в кабинет принеси…» ☺

Ребята, в конце важные пояснения:

Во-первых, кофе вам носить никто не будет, так что осторожнее в этом вопросе ☺

Во-вторых, когда сделаете работу – сильно не гордитесь. А лучше даже – никому не рассказывайте, что делали. Дело в том, что BAT-файлами сейчас уже никто не собирает… *«Вот жеж, падла!»* - говорите сейчас вы, и вы во многом правы. Но с чего-то надо начинать… Потерпите ☺

Однако, для особо нетерпеливых, вот чем собирают в Java-мире сейсас:

1. **Apache ANT** – <https://ant.apache.org> – на самом деле уже не собирают. ANT – абсолютно устаревший мамонт
2. **Apache Maven** – <https://maven.apache.org> – пока что все еще главный в Java-сборке
3. **Gradle** – <https://gradle.org> – Стремительно набирающий популярность сборщик. Возможно, со временем он даже вытеснит Maven, ровно как и Maven в свое время вытеснил ANT.