**Выпускной проект по StartJava —**

**”Книжный шкаф”**

* Всегда сопровождайте выводимые значения переменных или ввод с клавиатуры подходящим текстом
* Перед сдачей ДЗ сверьте свое решение со списком [рекомендаций](https://docs.google.com/document/d/1IAiDJmSAD86P-xTp3kOs-tFFuDQViXRewI-vpk1nT3g/edit#bookmark=id.h1bu5bp040sv)
* Присылайте на проверку только тот код, в котором нет замечаний [линтера](https://topjava.ru/blog/nastroyka-checkstyle-v-intellij-idea)

1. **ФУНКЦИОНАЛ**

1. Программа должна уметь:

* сохранять новую книгу
* по названию книги:
  + удалять ее
  + искать ее
* выводить количество (перед выводом шкафа):
  + всех книг
  + свободных полок
* выводить все книги (только для визуализации шкафа)
* очищать шкаф от книг
* предоставлять пользовательское меню

1. Книги сохраняйте по порядку, чтобы между ними не было пустых ячеек
2. При реализации удаления книгучитывайте, что после удаления между ними не должно быть промежутков. Используйте [System.arraycopy()](https://topjava.ru/blog/rukovodstvo-po-massivam-v-java-ch3#6)
3. Выводите подходящие сообщения, если книга не была *удалена*, *не* *найдена, не может быть сохранена* (в шкафу закончилось место или пользователь ввел недопустимую команду). При этом программа не должна ”падать” с ошибкой
4. Выводите подходящие сообщения после каждой удачно выполненной с помощью меню операции

1. **ИНТЕРФЕЙС**

1. При запуске программы выводите приветственный текст, отображая его с эффектом пишущей машинки

1. Для использования возможностей программы, выводите меню, например:

|  |
| --- |
| 1. Удалить книгу 2. Очистить шкаф 3. Завершить  или 4. save 5. delete 6. clear |

1. При вводе пользователем пункта меню не из списка, предоставляйте ему повторный ввод, а также выводите сообщение:

|  |
| --- |
| Ошибка: введите номер из списка: |

1. Выводите актуальное состояние шкафа после любой операции с книгами, а только затем — все не пустые полки, например:

|  |
| --- |
| В шкафу книг - 2, свободно полок - 8  |Ирвинг Стоун, Жажда жизни, 1973             | |--------------------------------------------| |Рэй Брэдбери, 451 градус по Фаренгейту, 1980| |--------------------------------------------| |                                            | |

1. Т. к. при запуске программы шкаф всегда пуст (или когда очищен), в этом случае выводите только меню и сообщение:

|  |
| --- |
| Шкаф пуст. Вы можете добавить в него первую книгу |

1. После каждой выполненной операции выводите сообщение:

|  |
| --- |
| Для продолжения работы нажмите клавишу <Enter> |

Только после нажатия *Enter* выводите шкаф и меню, т. к. у пользователя должна быть возможность увидеть результат работы команд.

1. **ИСПОЛЬЗУЕМЫЕ КЛАССЫ**

1. **Книга**:
   * Поля:

* автор
* название
* год издания

* Методы:
  + [toString()](https://www.examclouds.com/ru/java/java-core-russian/method-tostring)
  + прочие методы

1. **Книжный шкаф:**

* Поля:
  + количество книг в шкафу
  + массив для *десяти* книг
  + прочие поля

* Методы:
  + добавить книгу
  + найти и получить книгу
  + удалить книгу
  + (только для визуализации шкафа. Пользователям они недоступны):
    - получить все книги
    - получить количество книг в шкафу
    - получить количество свободных полок
  + очистить шкаф

1. **Класс для размещения:**
   * вывода меню
   * обработки выбора пользователя
   * ввода необходимой для методов класса *Книжный шкаф* информации
   * вывода возвращаемых этими методами данных

**Задание со \* (по желанию)**

* длина шкафа должна быть динамической и зависеть от длины самой длинной записи любой книги. При этом информация о книгах не должна выходить за пределы шкафа ни после удаления, ни после добавления книг
* используйте *Enum-значения* в *switch* при проверке выбора пользователем того или иного пункта меню. Текст для меню берите из полей *Enum*
* создайте собственные классы исключений для методов *save* и *delete* класса *Bookshelf*
* разместите классы исключений в папке *exception*

**Рекомендации**

1. Сдвиг книг выполняйте на исходном массиве без создания нового

1. Сдвигайте только книги, без *null*

1. Для вывода меню используйте [текстовые блоки](https://topjava.ru/blog/tekstovye-bloki-v-java)

1. Если метод возвращает результат своей работы, то выводите его в месте вызова данного метода, а не в самом методе

1. Методы в классе *Книжный шкаф* размещайте в порядке их следования в меню

1. Для отображения полок используйте [repeat()](https://topjava.ru/blog/java-11-string-api-additions#:~:text=%D1%81%C2%A0%D0%BD%D0%B8%D0%BC%D0%B8%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D1%82%D1%8C.-,2.%20repeat(),-%D0%9A%D0%B0%D0%BA%20%D1%81%D0%BB%D0%B5%D0%B4%D1%83%D0%B5%D1%82%20%D0%B8%D0%B7)

1. Пересчитывайте длину шкафа после удаления и сохранения книг только при потребности в ее изменении

1. Все неизменяемые значения должны быть константами: [1](https://topjava.ru/blog/rukovodstvo-po-modifikatoru-static-v-java#7), [2](https://refactoring.guru/ru/replace-magic-number-with-symbolic-constant)

**Ликбез**

Любую программу можно условно разделить на две части (слои приложения). На самом деле их больше, но сейчас это не важно (слои проходятся на нашей стажировке [TopJava](https://topjava.ru/topjava)):

* **бизнес-логика:** то, для чего создавалась программа (что-то подсчитать, найти, заменить и т. д.) — ее функциональные возможности
* **интерфейс:** то, что программа отображает в качестве результата своей работы, например, меню, подсчитанное значение, полученные данные, UI

Эти части (логика и отображение) в, идеале, не должны смешиваться.

Это связано с тем, что намного проще поддерживать код программы, когда известно, в каком из ее слоев возникла ошибка. Если программа неверно считает, то ошибку нужно искать в слое логики, если криво выводит результат — ошибка в слое отображения.

Это называется разделением зоны ответственности, когда каждый слой программы отвечает за свою часть работы.

В решаемых нами задачах хоть и нет графического интерфейса (кнопок, меню, тулбаров и т. д.), т. к. все программы написаны для консоли, но все же консоль — это тоже интерфейс, хоть и текстовый. Следуя этой логики, в идеале, мы должны были бы создать новый класс, отвечающий только за вывод чего-либо на консоль. Но для простоты, мы этого не делаем (если хотите, можете попробовать). Единственное, что мы можем сделать в заданных рамках — это осуществлять вывод в консоль в том месте программы, где был получен какой-то результат от вызванного метода, а не в самом методе.

**Например:**

* информацию о найденной книге следует выводить не в методе *find*, а в вызываемом метод классе
* сгенерированное число, как вы помните, не выводится в классе *Random*, а возвращается в виде значения, которое уже мы можем выводить
* а вот если нужно очистить массив с числами в классе *Player*, то метод *clear()* возвращает *void*, а значит все, что он может — это вывести сообщение, что числа очищены только внутри себя

**Но есть нюансы.**

Например, вам нужно вывести текст ошибки, которая возникла в методе. Где ее выводить: в методе или в том классе, где этот метод вызывался? Если вы используете обычный *System.out.println("Текст ошибки"),* и при этом метод возвращает *void*, то выводить сообщение нужно в самом методе, т. к. тут без вариантов (хотя по-хорошему, нужно вместо *sout* использовать логирование, но мы его пока не применяем).

Если вы используете систему исключений, то сообщение необходимо указывать в конструкторе иксепшена, например, *throw new RuntimeException("Текст ошибки")*. Затем в классе, в котором этот иксепшен ловится через *catch*, выводить сообщение через *System.out.println(e.getMessage()).* Это позволит для одинаковых иксепшенов выводить разный текст.