**КУРСОВОЙ ПРОЕКТ**

**МНОГОПОТОЧНАЯ МОДЕЛЬ ОБСЛУЖИВАНИЯ ПОТОКА ЗАЯВОК НА РАЗГРУЗКУ, ПОСТУПАЮЩИХ ОТ ГРУЗОВЫХ СУДОВ, ПРИБЫВАЮЩИХ В МОРСКОЙ ПОРТ**

по дисциплине «Объектно-ориентированное программирование»

Выполнил

студент гр. 3530904/90001 <подпись> Степанов А. А.

Руководитель <подпись> Эйзенах Д. С.

# Оглавление

[Оглавление 2](#_Toc72515672)

[Постановка задачи 2](#_Toc72515673)

[Реализация 4](#_Toc72515674)

[Сервис 1 5](#_Toc72515675)

[Сервис 2 7](#_Toc72515676)

[Сервис 3 8](#_Toc72515677)

[Результаты работы 15](#_Toc72515678)

# Постановка задачи

**Сервис 1**

Реализовать генератор расписания прибытия судов. Расписание включает:

* день и время прибытия
* название судна
* вид груза (сыпучий, жидкий, контейнер) и его вес в тоннах или штуках (для контейнеров)
* планируемый срок стоянки в порту для разгрузки - вычисляется на основе веса груза и производительности крана

**Сервис 2**

Получает данные из сервиса 1 и сервиса 3 и сохраняет их в json-файл. Предусмотреть возможность ручного добавления записей через консоль.

**Сервис 3**

Грузовые суда прибывают в порт согласно расписанию (порт работает круглосуточно), но возможны опоздания и досрочные прибытия.

Для разгрузки судов в порту используются три вида разгрузочных кранов, соответствующих трем видам грузов: сыпучим и жидким грузам, контейнерам. Число разгрузочных кранов каждого вида ограничено (изначально всех кранов по 1), так что поступающие заявки на разгрузку одного вида груза образуют очередь. Длительность разгрузки судна зависит от вида и веса его груза. Каждый кран работает в отдельном потоке, одно судно могут разгружать не более двух кранов одновременно.

Любой дополнительный (сверх запланированного срока) час стояния судна в порту (из-за ожидания разгрузки в очереди или из-за задержки самой разгрузки) влечет за собой выплату штрафа 100 у. е. за каждый час простоя судна.

При моделировании прибытия судов отклонение их от расписания рассматривается как случайная величина в интервале от -7 до 7 дней. Еще одной случайной величиной, изменяющейся в диапазоне от 0 до 1440 минут, является время задержки окончания разгрузки судна по сравнению с обычным (зависящим только от вида груза и его веса).

Цель моделирования работы морского порта – определение для заданного расписания прибытия судов минимально достаточного числа кранов в порту, позволяющего минимизировать штрафные суммы при стоимости крана в 30 000 у. е. Период моделирования – 30 дней. В параметры моделирования следует включить расписание прибытия судов и количество кранов каждого вида.

В результате работы программы должен быть сформирован отчёт, содержащий: список произведенных разгрузок, в котором указывается название загруженного судна, время его прихода в порт и время ожидания в очереди на разгрузку (в формате дд:чч:мм), время начала разгрузки и ее продолжительность, а также по окончании моделирования должна быть выведена итоговая статистика: число разгруженных судов, средняя длина очереди на разгрузку, среднее время ожидания в очереди, максимальная и средняя задержка разгрузки, общая сумма штрафа и итоговое необходимое количество кранов каждого вида.

**Этапы реализации**

Этап 1:

* Сервис 1: результат выводить в консоль.
* Сервис 2: вызывает метод генерации, описанный в сервисе 1; получает на вход метода данные от сервиса 3
* Сервис 3:
* используется json-файл, полученный в результате работы сервиса 2;
* результат работы выводится в консоль.

Этап 2:

* Сервис 1: GET - эндпоинт возвращающий расписание.
* Сервис 2: обращается GET - запросом к соответствующему эндпоинту. Предоставляет следующие эндпоинты:
* GET - эндпоинт для получения расписания в виде json-документа;
* GET - эндпоинт, возвращающий расписание по имени json-файла или ошибку, если такого файла нет;
* POST-эндпоинт для сохранения результатов работы сервиса 3 в json-документ.
* Сервис 3:
* обращается к GET - эндпоинту сервиса 2 для получения расписания в виде json-файла
* результат работы отправляется на POST-эндпоинт сервиса 2.

# Реализация

Программа работает в трех разных проектах одновременно. Каждый проект представляет из себя один сервис из представленных в задании. Пользователь может работать только с сервисом №2: пользователь отправляет определенный запрос сервису №2, после чего данный сервис отправляет запросы другим сервисам или работает с сохраненными данными.

Сервис №2 предоставляет пользователю следующие эндпоинты:

1. @PostMapping("/service2/create-string-schedule")  
   public ResponseEntity<String> getScheduleForUser(@RequestParam("shipnumber") int numberOfShipsToStore,  
    @RequestParam("min-weight") double minWeight,  
    @RequestParam("max-weight") double maxWeight)

Данный эндпоинт ведет к созданию расписания кораблей с заданными количеством кораблей, минимальным и максимальным весами груза, его сохранению в json-файл и его возвращению пользователю в формате строки.

1. @GetMapping("/service2/get-string-schedule")  
   public ResponseEntity<String> getScheduleForUser()

Данный эндпоинт возвращает расписание в виде строки, если само расписание было создано. Если же нет, то выбрасывается **IOException**.

1. @GetMapping("/service2/get-statistics")  
   public ResponseEntity<String> getStatistics()

Данный эндпоинт ведет к моделированию составленного расписания, его сохранению в json-файл и его возвращению пользователю в формате строки. Если расписание не было создано, выбрасывается **IOException**.

1. @PostMapping("/service2/post-new-ship")  
   public ResponseEntity<HttpStatus> postNewShip(@RequestBody JSONObject jsonShip)

Данный эндпоинт позволяет внести новый корабль в расписание. Если расписание не было создано, выбрасывается **IOException**. Телом запроса служит json-объект, в котором содержится имя корабля (**“name”**), вид груза (**“cargoType”**), вес груза (**“weight”**) и время прибытия (**“arrivingTime”**).

Далее разберем функционал всех сервисов и их эндпоинтов.

## Сервис 1

Сервис 1 предоставляет только 1 эндпоинт:

@GetMapping("/service1/get-schedule")  
public ResponseEntity<String> getSchedule(@RequestParam("shipnumber") int numberOfShipsToStore,  
 @RequestParam("min-weight") double minWeight,  
 @RequestParam("max-weight") double maxWeight)

Данный эндпоинт создает расписание на основе полученных данных, преобразует его в строку и возвращает ее.

Все классы и перечисления, созданные для сервиса 1 служат и для других сервисов в качестве вспомогательных структур. Для определения вида груза существует перечисление **CargoType**:

public enum CargoType {  
 *CONTAINER*,  
 *LOOSE*,  
 *LIQUID*}

Для работы со временем существует класс **Time**, хранящий значения дня, часа, минуты, а также статические значения максимально возможного числа дней, часов и минут. **Time** хранит следующие поля:

private long day;  
private long hour;  
private long minute;  
  
public static final int *maxDay* = 30;  
public static final int *maxHour* = 23;  
public static final int *maxMinute* = 59;

**Time** реализует следующие конструкторы:

* public Time(long day, long hour, long minute)

public Time(Time time)

public Time(String string)

Конструктор с аргументом в виде строки ожидает на вход время в формате “дд:чч:мм”. **Time** также реализует следующие методы:

* public long getDay()  
  public long getHour()  
  public long getMinute()

Стандартные геттеры для получения значений полей.

* public void makeEqual(Time time)

Приравнивает время к времени, переданному в аргументах.

* public int compareTo(Time time)

Сравнивает два разных времени.

* public boolean equals(Time time)

Возвращает true, если два времени равны и false, если наоборот.

* public void addMinutes(double minute)

Добавляет ко времени заданное количество минут.

* public long getTimeInMinutes()

Возвращает время в минутах.

* public static Time getRandomTime(int numberOfDays)

Возвращает случайное время в промежутке от нулевого до заданного номера дня.

* public static String toString(Time time)

Возвращает время в формате строки.

Для группировки данных груза в один объект был написан класс **ScheduleElement**. Он содержит в себе имя пришедшего корабля, время его прибытия, вид и вес груза, а также время на его разгрузку. **ScheduleElement** реализует следующие конструкторы:

* public ScheduleElement(Time arrivingTime, String name, CargoType cargoType, double weight)

public ScheduleElement(ScheduleElement scheduleElement)

**ScheduleElement** также реализует следующие методы:

* public Time getArrivingTime()  
  public String getName()  
  public CargoType getCargoType()  
  public double getWeight()  
  public Time getUnloadingTime()

Стандартные геттеры для получения значений полей.

Для создания расписания, состоящего из **ScheduleElement** был создан класс **Schedule**. **Schedule** содержит в себе следующие поля:

private int numberOfShipsToStore;  
private double minWeight;  
private double maxWeight;  
private int numberOfShipsInSchedule = 0;  
private String beginningOfShipName = "Ship ";

**Schedule** также содержит в себе **LinkedList** для хранения всех **ScheduleElement**, а также компаратор для сравнения двух элементов по времени прибытия корабля. **Schedule** реализует следующий конструктор:

public Schedule(int numberOfShipsToStore, double minWeight, double maxWeight)

**Schedule** так же реализует следующие методы:

* public LinkedList<ScheduleElement> getScheduleElementList()

Стандартный геттер для получения списка **ScheduleElement**.

* public static String toString(Schedule schedule)

Статический метод, возвращающий преобразование расписания в строку.

* public static String toString(LinkedList<ScheduleElement> scheduleList)

Статический метод, возвращающий преобразование списка **ScheduleElement** в строку.

* public static LinkedList<ScheduleElement> getScheduleListFromString(String string)

Статический метод, возвращающий преобразование строки в список **ScheduleElement**.

## Сервис 2

Сервис 2 предоставляет следующие эндпоинты помимо пользовательских:

* @GetMapping("/service2/get-json-schedule-by-name")  
  public ResponseEntity<String> getScheduleByName(@RequestParam("file-name") String fileName)

Данный эндпоинт возвращает расписание в виде строки по имени файла, в котором оно должно храниться. Если такого файла нет, выбрасывается **IOException**.

* @PostMapping(value = "/service2/post-statistics")  
  public ResponseEntity<HttpStatus> postStatistics(@RequestBody String statistics)

Данный эндпоинт сохраняет полученную в виде строки статистику моделирования в json-файл.

Для работы с json-файлами и json-объектами в сервисе 2 существует класс JSONService. Этот класс не имеет конструкторов и состоит целиком из статических методов:

* public static LinkedList<ScheduleElement> getScheduleListFromJSON(JSONObject schedule)

Статический метод, извлекающий список **ScheduleElement** из json-объекта и возвращающий его.

* public static JSONObject toJSON(LinkedList<ScheduleElement> scheduleList)

Статический метод, сохраняющий список **ScheduleElement** в json-объект и возвращающий его.

* public static String JSONToModelString(JSONObject obj)

Статический метод, извлекающий строку со статистикойиз json-объекта и возвращающий ее.

* public static JSONObject modelStringToJSON(String string)

Статический метод, сохраняющий строку со статистикойв json-объект и возвращающий его.

* public static JSONObject toJSON(ScheduleElement scheduleElement)

Статический метод, сохраняющий **ScheduleElement** в json-объект и возвращающий его.

* private static JSONObject toJSON(Time time)

Статический метод, сохраняющий **Time** в json-объект и возвращающий его.

* private static Time getTimeFromJSON(JSONObject obj)

Статический метод, извлекающий времяиз json-объекта и возвращающий его.

## Сервис 3

Сервис 3 предоставляет следующий эндпоинт:

@PostMapping(value = "/service3/create-statistics")  
public ResponseEntity<HttpStatus> createAndSendStatistics(@RequestBody JSONObject nothing)

Данный эндпоинт получает расписание из 2 сервиса, моделирует его и отправляет строку со статистикой 2 сервису для создания с ней json-файла. В сервисе 3 элементы расписания хранятся в объектах класса **ScheduleElementKeeper**, с помощью которого удобно отображать разгрузку грузов. **ScheduleElementKeeper** содержит в себе следующие поля:

private boolean isFinished;  
private ScheduleElement scheduleElement;  
private int numberOfCranes;  
private long minutesUnloaded;  
private long minutesForUnloading;  
private Time actualArrivingTime;  
private Time startUnloadingTime;  
private Time finishUnloadingTime;

Поле **numberOfCranes** отображает, сколько кранов разгружают груз в данный момент, а поле **isFinished** показывает, закончена ли разгрузка. **ScheduleElementKeeper** реализует следующие конструкторы:

* public ScheduleElementKeeper(ScheduleElement scheduleElement)
* public ScheduleElementKeeper(ScheduleElementKeeper cargo)

**ScheduleElementKeeper** реализует следующие методы:

* public void finish()

Присваивает переменной **isFinished** значение true.

* public void addCrane()

Увеличивает количество кранов, которые разгружают груз в данный момент на 1.

* public void addMinutesUnloaded(long minutes)   
  public void addMinutesForUnloading(long minutes)

Прибавляют соответствующим полям со временем заданное количество минут.

* public void setActualArrivingTime(Time time)  
  public void setStartUnloadingTime(Time time)   
  public void setFinishUnloadingTime(Time time)

Стандартные сеттеры, присваивающие соответствующим полям аргументы методов.

* public ScheduleElement getScheduleElement()  
  public int getNumberOfCranes()  
  public String getName()

public long getMinutesUnloaded()  
 public long getMinutesForUnloading()  
 public Time getActualArrivingTime()  
 public Time getStartUnloadingTime()  
 public Time getFinishUnloadingTime()

public boolean isFinished()

Стандартные геттеры, возвращающие значения соответствующих полей.

* public void checkMinutesForUnloading()

Присваивает **minutesForUnloaging** значение в минутах необходимого для разгрузки времени из **scheduleElement**.

* public Time getWaitForStartOfUnloadingTime()

Возвращает время, которое корабль ждал в порту до начала разгрузки.

* public Time getUnloadingDuration()

Возвращает время разгрузки.

В сервисе 3 краны моделируются классом **Crane**. **Crane** содержит в себе следующие поля:

* volatile private boolean isActive;  
  volatile private boolean isBusy;  
  volatile private boolean changeTimeEvent;

Флаги, которые показывают соответственно запущен ли поток крана, разгружает ли кран какой-либо груз и происходит ли какое-нибудь событие в данный момент. “Событием” далее будем называть момент полной разгрузки определенного груза или момент прихода определенного судна в порт.

* private Time previousModelTime;

Переменная времени, с помощью которой можно понять, сколько прошло времени с предыдущего события.

* private ScheduleElementKeeper cargo

Объект **ScheduleElementKeeper** является грузом, который кран разгружает в данный момент.

**Crane** реализует конструктор по умолчанию, а также следующие методы:

* public void run()

Запускает поток крана.

* public boolean isActive()  
  public boolean hasTimeEvent()  
  public boolean isCraneBusy()

Стандартные геттеры для получения значений соответствующих полей.

* public void disable()

Завершает поток крана.

* public void createChangeTimeEvent()

Показывает крану, что наступило событие.

* public void getCargoForCrane(ScheduleElementKeeper scheduleElementKeeper)

Присваивает **cargo ScheduleElementKeeper**, переданный в качестве аргумента.

* private synchronized void unloadCargo()

Метод, который высчитывает, какую работу совершил кран в промежутке между **previousTime** и настоящим временем и отображающий это в **cargo**.

Класс **Crane** является частью класса **CargoThread**, симулирующего процесс разгрузки. **CargoThread** содержит в себе компаратор по сравнению элементов расписания по их времени прибытия, а также следующие поля:

public final static int *maxUnloadingMinutesDelay* = 1440;  
public final static int *arrivingDaysDeviationWindow* = 14;  
private Time modelTime = new Time(1, 0, 0);  
private LinkedList<ScheduleElementKeeper> arrivingCargoList;  
private LinkedList<ScheduleElementKeeper> arrivedCargoList;  
private LinkedList<ScheduleElementKeeper> unloadedCargoList;  
private Crane[] arrayOfCranes;  
private int numberOfCranes;  
private int queueLength;  
private int numberOfQueueEvents;

**CargoThread** реализует следующий конструктор:

public CargoThread(LinkedList<ScheduleElementKeeper> arrivingCargoList, int numberOfCranes)

**CargoThread** реализует следующие методы:

* public void run()

Запускает поток очереди.

* public LinkedList<ScheduleElementKeeper> getArrivedCargoList()  
  public LinkedList<ScheduleElementKeeper> getUnloadedCargoList()  
  public double getQueueLength()  
  public int getNumberOfQueueEvents()

Стандартные геттеры для получения значений соответствующих полей.

* private void addTimeDeviations(ScheduleElementKeeper cargo)

Добавляет временные отклонения к времени прибытия груза и времени разгрузки.

* private boolean isEveryCraneActive()

Проверяет, запустились ли все необходимые потоки кранов.

* private boolean isEveryCraneDisabled()

Проверяет, завершились ли все потоки кранов.

* private boolean didEveryCraneMakeUnloading()

Проверяет, все ли краны произвели разгрузку во время события.

* private double getNearestEventInMinutes()

Возвращает количество минут до следующего события.

* private void getArrivingCargosIntoArrivedList()

Переносит прибывшие по времени грузы из списка прибывающих в список прибывших грузов.

* private void createChangeTimeEvent()

Создает событие.

* private void distributeCranesBetweenArrivedCargos()

Распределяет свободные краны между прибывшими грузами.

* private Crane getUnbusyCraneForWork()

Возвращает кран, который ничего не разгружает на данный момент.

* private int countBusyCranes()

Считает количество кранов, которые разгружают грузы в данный момент.

* private void updateConditionOfFinishedCargos()

Переносит разгруженные грузы из списка прибывших в список разгруженных.

* private int countUnbusyShips()

Возвращает количество кораблей, которые не разгружаются в данный момент.

**CargoThread** создается и много раз запускается внутри класса **StatisticsGetter**. Данный класс подсчитывает статистику для грузов одного вида. **StatisticsGetter** содержит в себе следующие поля, которые как раз и представляют из себя общую статистику моделирования:

public final static int *numberOfCargoThreadLaunches* = 1000;  
public final static int *additionalCraneFine* = 30000;  
public final static int *hourFine* = 100;  
public final static int *defaultNumberOfCranes* = 1;  
private LinkedList<ScheduleElementKeeper> listForCargoThread;  
private LinkedList<ScheduleElementKeeper> finalListOfUnloadedCargos;  
private CargoThread cargoThread;  
private int numberOfCranes;  
private double fineValue;  
private int queueLength;  
private int numberOfQueueEvents;

**StatisticsGetter** реализует следующий конструктор:

public StatisticsGetter(LinkedList<ScheduleElementKeeper> list)

**StatisticsGetter** реализует следующие методы:

* public void run()

Запускает поток сборщика статистики.

* public double getOverallFine()  
  public int getOverallQueueLength()  
  public int getOverallNumberOfQueueEvents()  
  public int getFinalNumberOfCranes()  
  public LinkedList<ScheduleElementKeeper> getListForCargoThread()

Стандартные геттеры для получения значений соответствующих полей.

* private double getFineValue(CargoThread cargoThread)

Возвращает штраф, полученный в заданном **CargoThread.**

* private long getFineValueFromCranes(int numberOfCranes)

Возвращает стоимость используемых кранов.

**StatisticsGetter** запускается внутри класса **ModelPreparer**, который собирает всю статистику вместе. **ModelPreparer** содержит в себе компаратор для сравнения элементов расписания по времени прибытия, а также следующие поля:

private LinkedList<ScheduleElement> listForJSON;  
private LinkedList<ScheduleElementKeeper> containerList;  
private LinkedList<ScheduleElementKeeper> looseList;  
private LinkedList<ScheduleElementKeeper> liquidList;  
private LinkedList<ScheduleElementKeeper> overallListOfUnloadedShips;  
private int overallNumberOfUnloadedShips;  
private int overallQueueLength;  
private int overallNumberOfQueueEvents;  
private double averageQueueLength;  
private double overallFineValue;  
private int requiredNumberOfContainerCranes;  
private int requiredNumberOfLooseCranes;  
private int requiredNumberOfLiquidCranes;  
private double averageQueueWaitInMinutes;  
private double averageUnloadingDelayInMinutes;  
private long maxUnloadingDelayInMinutes;

**ModelPreparer** реализует следующий конструктор:

public ModelPreparer(LinkedList<ScheduleElement> scheduleList)

**ModelPreparer** реализует следующие методы:

* public LinkedList<ScheduleElementKeeper> getOverallListOfUnloadedShips()  
  public int getOverallNumberOfUnloadedShips()  
  public double getAverageQueueLength()  
  public double getOverallFineValue()  
  public int getRequiredNumberOfContainerCranes()  
  public int getRequiredNumberOfLooseCranes()  
  public int getRequiredNumberOfLiquidCranes()  
  public double getAverageQueueWaitInHours()  
  public double getAverageUnloadingDelayInHours()  
  public double getMaxUnloadingDelayInHours()

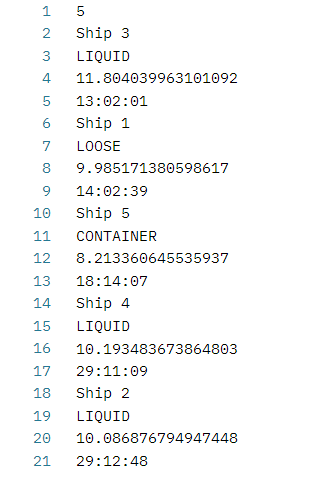
Стандартные геттеры для получения значений соответствующих полей.

* public static String toString(ModelPreparer modelPreparer)

Статический метод, преобразующий всю статистику в строку и возвращающий ее.

# Результаты работы

Отправим работающей программе следующий запрос: <http://localhost:8082/service2/create-string-schedule?shipnumber=5&min-weight=7&max-weight=12>. После этого мы получаем само расписание:



Теперь отправим программе следующий запрос: <http://localhost:8082/service2/get-statistics>. После этого получаем статистику моделирования:

