Task Manager

Development of an optimised scheduling tool

Group number: S2126631638

Members: Anna Brovko (68881)
Hamza Yalmaz Mahmood (68898)
Lisbeth Amalie Collin (69228)

Supervisor: Line Reinhardt

Date: 2/6-2021

Characters: 56.572

Abstract

This project examines how to develop a tool for task scheduling using the Google Calendar API. Our program aims to be helpful for students who have difficulties planning the school assignments into their calendar.

We use *UML diagrams* to visualize the functionalities of our application and create a structure of the program using the *use case* and *class diagram,* respectively. Along with the *UML diagrams,* we use Java Collection methods to sort the user assignments and send them to the calendar through API calls between our application and the Google Calendar. Furthermore, we apply the *mergeSort* algorithm that comes with the Collections sorting method. We also create some rules for the sorting process and the insertion of tasks into the calendar.

Using the mentioned tools and methods, we have integrated the Google Calendar and our application, which inserts the tasks into the user's calendar. The scheduling algorithm handles the tasks' deadline and duration, splits them into multiple events and sends them to the free time slots.

Table of Contents

Introduction

During our time in the university, we have discovered that sometimes it can be challenging to schedule school-related activities into a calendar. Therefore, our project idea is to create an algorithm that will optimize task management by creating events out of the assigned tasks and placing them into suitable spots in the calendar. This scheduling algorithm will automate the planning of school assignments and save users' time finding time slots in the calendar and calculating the hours for each task. It will consider several aspects before scheduling the tasks into the calendar.

To achieve the goal, we will implement algorithms that will sort the assignments, split them into events, if needed, and send them to the available slots in the Google Calendar connected to our program. Properties for each task, such as priority, deadline, and duration, will be considered when organizing the entered tasks before sending the task events to the calendar. For that, we will create rules for our program. We will dive into the area we want to research and which tools we use to achieve our goal.

Research area

In this project, our research will focus on the technical aspects of how to develop an app that will help users to insert events with tasks they need to accomplish into a Google Calendar. Therefore, we will investigate how to start a Java project using Google Calendar API. Thus, we will also explore how to send events from our program to a Google Calendar using the Google Calendar API. In this project, we will find out how to sort the tasks by pre-defined criteria, create events for each assignment and send them to the calendar.

Research questions

Based on the research area, we have prepared a problem definition, which we will work on this semester:

How can a scheduling algorithm be implemented, and how can we use the Google Calendar API to insert tasks generated in a program into a Google Calendar?

To answer our research question, we have also formulated some sub-questions which we will use to give structure to our report and code:

1. **How can a program be constructed to schedule events/tasks in a personal calendar?**
   We will answer this question by first creating a Use Case diagram to visualize our motivation with the program. Secondly, we will create a Class Diagram with the classes and methods we need to structure our program. We will also select a development environment with the tools essential to reach the goal, such as a programming language, code editor and version control system.

   a. **How can the Google Calendar API be used to schedule tasks in a calendar?**
      The Google Calendar API gives us access to the personal Google Account, from where we can retrieve the events. We will examine how the Google Calendar API works and how we can integrate it into our code. We will also find out how we can benefit from using its interface.

   b. **How to design a scheduling algorithm in terms of scheduling tasks most optimally?**
      We will also learn the JodaTime package that we can use to manage the time and date entities. We will create a set of rules and criteria for sorting the tasks most optimally. Here we will use the Java Collections framework and send the sorting instructions. The program will send the tasks to the calendar in an order obtained after sorting.

   c. **What is a greedy algorithm, and how can it be implemented in our program?**
      We will look at the greedy algorithm logic and how we can use it, e.g., to find the available spots in the program and send events into the first available free slots in the calendar.

Project boundaries

Our objective with the program is to have a working application that will organize tasks and make task management simpler. We have decided to postpone front-end development for our program and focused on the back-end. Though by using the Google Calendar API, we partly achieve the front-end. It allows us only to focus on the back-end code.

For sorting the tasks, we implement the Java Collection methods and algorithms to limit the number of new ones that we would have to create. With access to Java Collections, we can use the `Collections.sort()`, which has all the functionality we need. It chooses the sorting algorithm according to the given data structure.

Another delimitation for our program will be a set of rules that we implement in our program, which we will explain in this report. Though, the program needs more extended rules for different scenarios. For the next iteration, we can develop additional ones and implement them in the program. We have in the current phase selected the rules that are possible to accomplish within the time limit for this project.

Moreover, we have simplified the code, so it only asks for the hour entities when creating the work time interval. Also, the deadline of the task is limited to the date without the time entity. That means the user will only be able to create work time intervals within the complete hours. So, the user can provide the work interval between 9 and 15, but not between 9:15 and 15:30. We will explain this approach later in the report.

In the next section, we will dive into how we use the methods and tools we have chosen to deliver a working scheduling program.

Methodology and tools

We use Java as a programming language, as it follows the concepts of object-oriented programming (OOP). The OOP principles, such as inheritance and polymorphism, let us re-use the methods on different occasions. In this section, we will look closer at the technologies we use in program development. We will explain how we use the UML (Unified Modelling Language) diagrams and the version control tool to follow up with the changes made in the code.

Use Case diagram

The Use Case Diagram shows the purposes of our program and its functions. In our Use Case diagram, we have two actors – Student and Google Calendar. The student is the primary actor who is always needed to launch the application and interact with it. The Google Calendar is a secondary actor, which will interact with the Task Manager only when there is a need to exchange data between the Task Manager and Google Calendar.

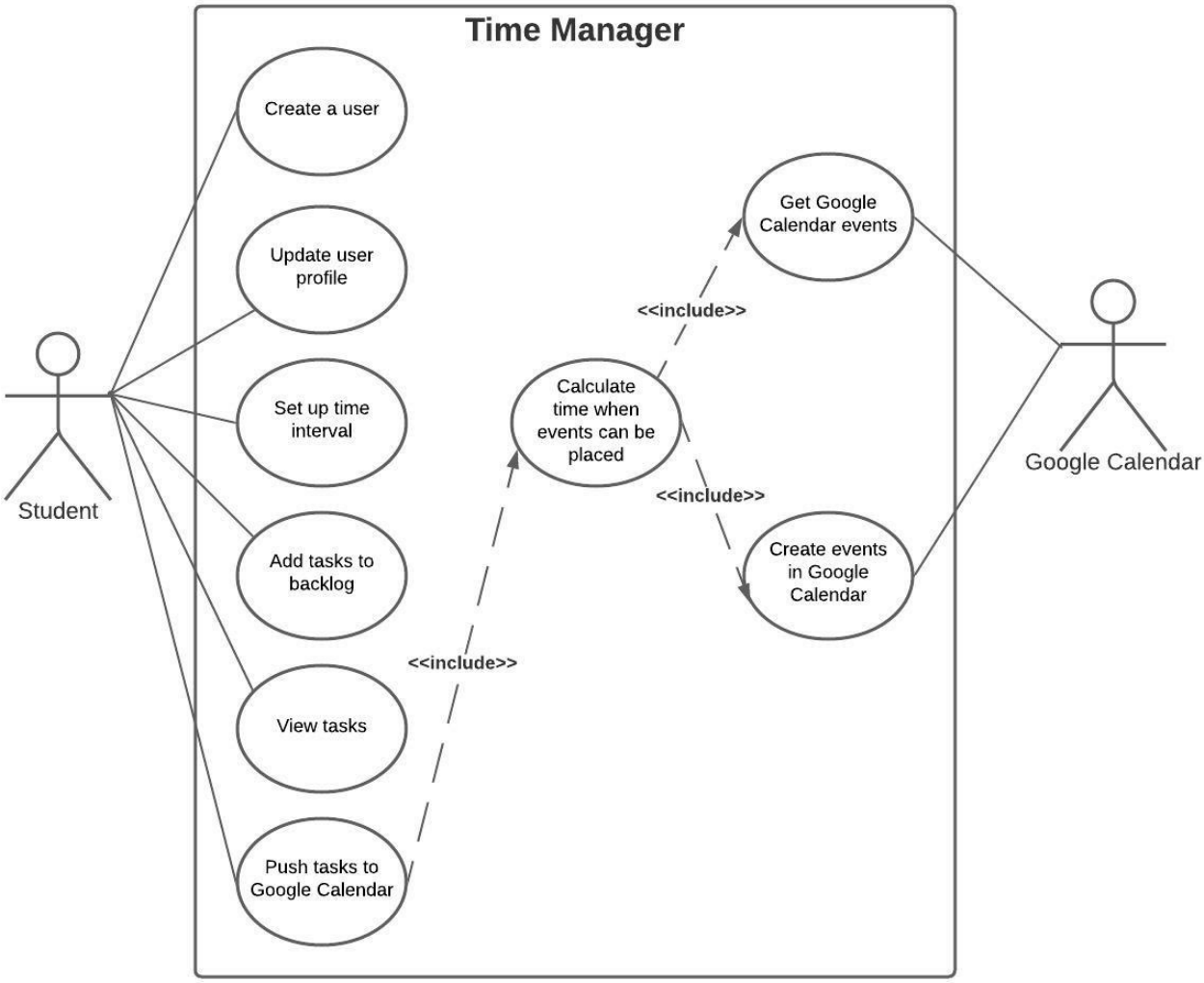


Figure 1. Use Case Diagram

In our diagram, we add the main actions that the user can do in the Task Manager application. According to it, the student should have the possibility to create a user, update the profile, set up work time interval when the tasks can be scheduled, add the tasks along with their properties. The function to send the entered tasks into the

calendar includes further actions that require interaction with the Google Calendar by sending the API calls. The Task Manager needs to find all the available slots in the calendar from tomorrow until the date the assignment should be completed or submitted. For this process, the Task Manager requires information from the Google Calendar, i.e., receiving all the events from the Google Calendar in the given time range and then sending the newly created events to the Google Calendar to the first available time slots in the calendar within the work time interval.

Class Diagram

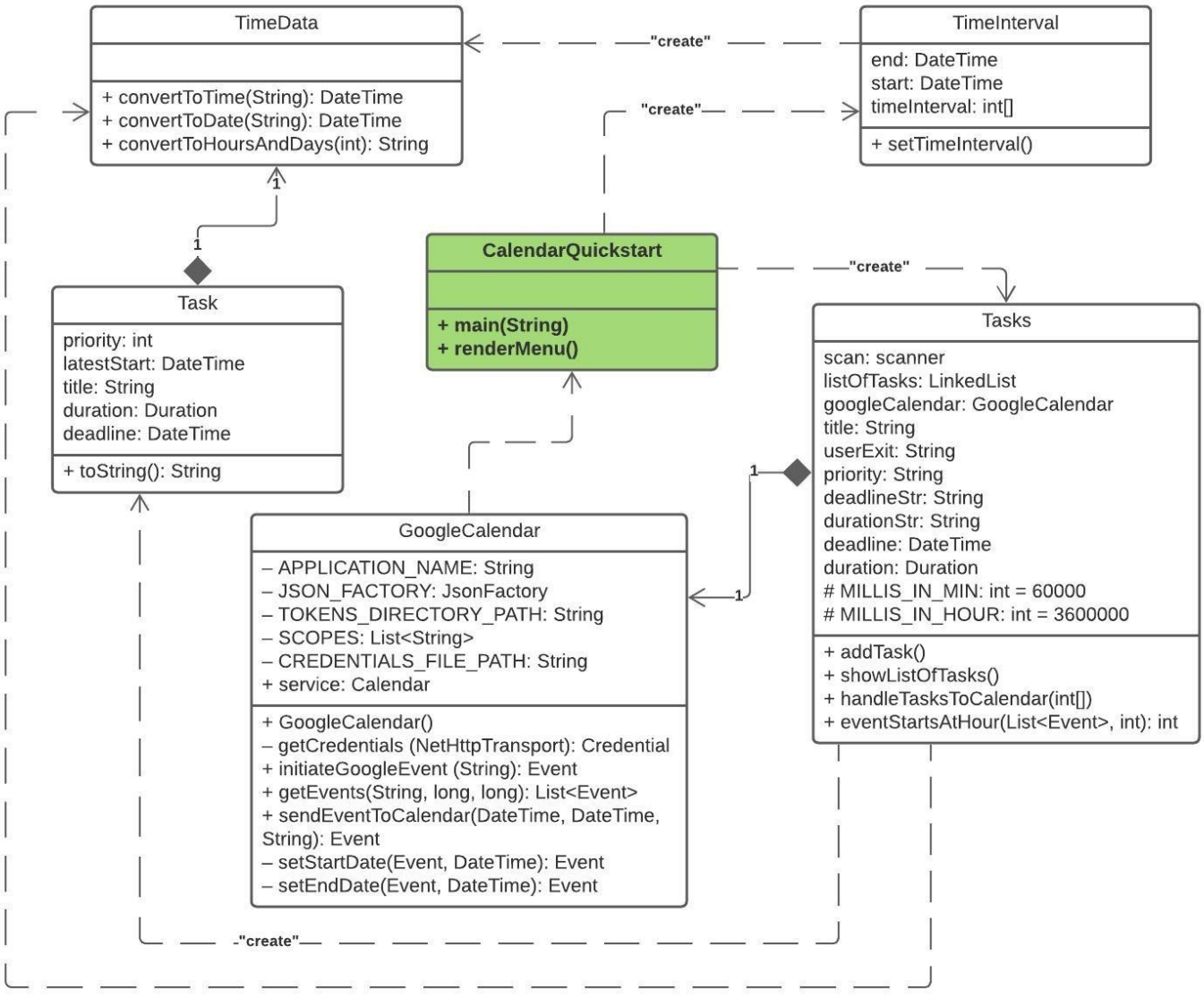


Figure 2. Class Diagram

The CalendarQuickstart class has the main method, where the program renders a menu. Here the user interacts with the menu and chooses the next method to be executed. From this class, the user can execute a method to create a work time interval. The method `setTimeInterval()` is placed in the class TimeInterval. When inserting the work time interval, the user input is getting formatted to be suitable for the time data, and for that, we use methods from the TimeData class. Here we create functions `convertToTime()`, `convertToDate()` and `convertToDaysAndHours()`, which we use on different occasions in other classes, such as Tasks and Task. In our class diagram, we have few composition relationships. The TimeData class depends on the Task class, where the multiplicity notation "1" in both directions means that for one instance of task, there should be one instance of TimeData. The same applies to the GoogleCalendar class, which is dependent on the Tasks class.

Version Control Systems (VCS)

When developing a program, it is beneficial to follow up with the changes and versions of the code we write. As we are a group of people working on the same project, we need to share and merge the latest changes made in the source code. Version control is a system for developing software that records and creates a history of changes that have been made to a file or set of files, making it possible to restore one of the previous versions of a file, if needed.

Centralized Version Control Systems (CVCS)

The version control which we are using in our project is a centralized version control system. It was developed to collaborate and deal with the versions of the file that developers commit changes to. The CVCSs have a single server that keeps all the versioned files, and the clients with access to it can check out files from the version database. The CVCS is easy to administer due to its access control administration (Git, u.d.).

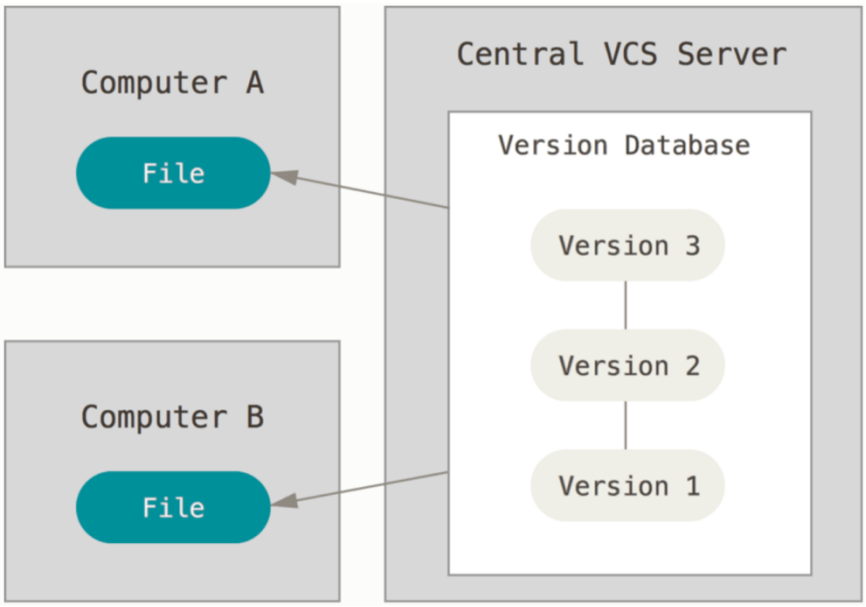


Figure 3. Centralized version control (Git, u.d.)

In our project, we need to control the different versions or changes in the source code. Therefore, we have chosen the version control software called Git and the platform GitHub. GitHub is a provider of Internet hosting using Git, and we were introduced to it during our classes. It has a user-friendly interface and provides collaboration features, such as project and task management and bug tracking. We have created a repository to save the versions of our code. It shows the timeline with an overview of what has been done to the code, which we use to go back to previous versions and look for the lines of code that we can reuse.

Figure 4. Git in IntelliJ

Besides, we are using an integrated development environment (IDE) called IntelliJ by JetBrains, from where we can interact directly with our Git repository with the source code. It is possible to commit and push changes to Git within IntelliJ, though we usually use the GitHub Desktop App, where we can see the edited files, see changes in these files and commit them to a chosen branch.

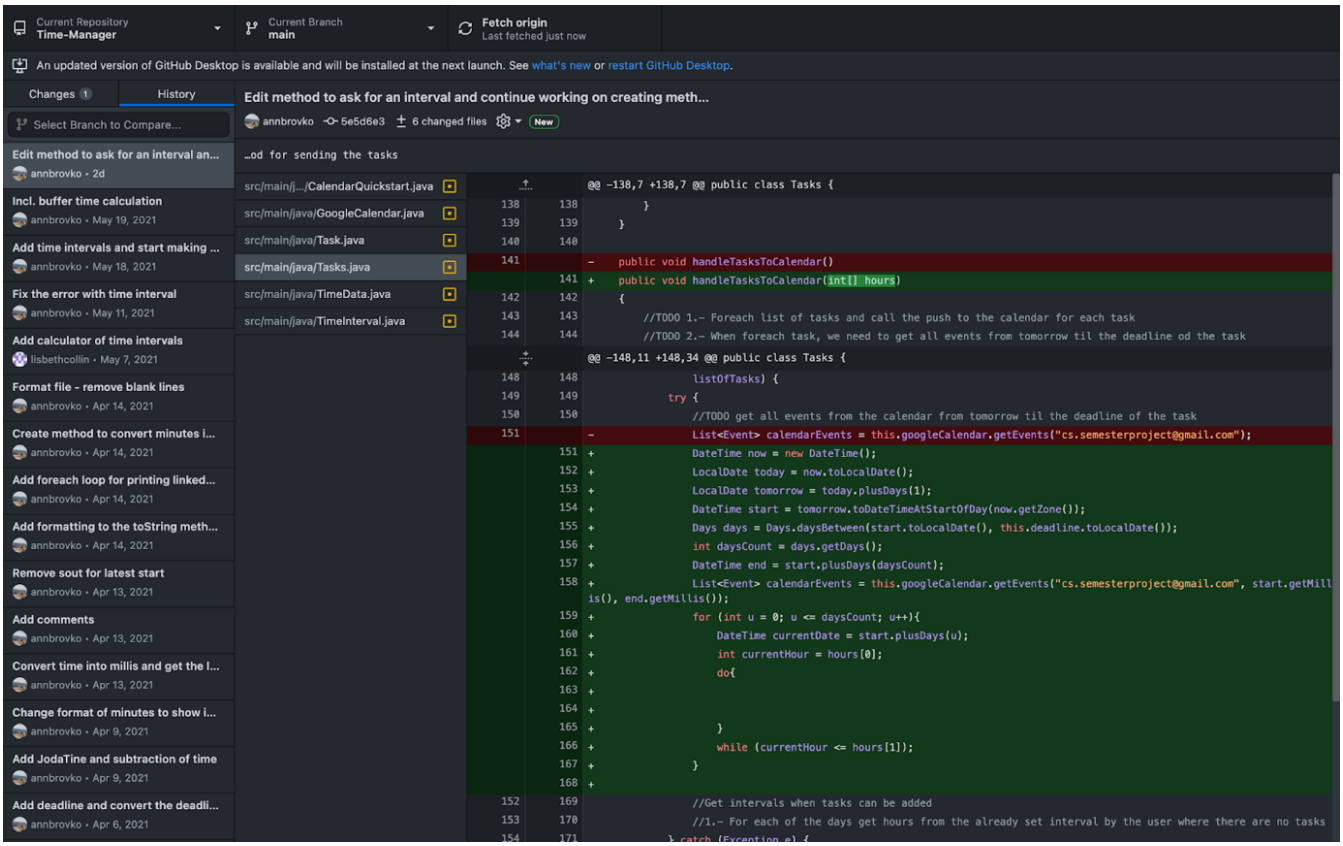


Figure 5. GitHub Desktop application

However, running Git commands inside of an IDE can be advantageous because we don't need to use the GitHub Desktop application, which makes the workflow faster and minimizes the number of steps from making a change to a code to pushing it to the repository.

Greedy algorithm

The greedy algorithm is an algorithmic paradigm used in optimization problems, i.e., finding the best possible solution to a given problem out of all possible ways to solve it. The greedy approach is helpful when finding locally optimal solutions as it tends to find optimal solutions to subproblems one by one. Though, it does not always find the globally optimal solution. The shortest/largest path problem is an example of the greedy algorithm that does not ensure the optimal solution, where it gets solved with a dynamic programming algorithm instead. The greedy algorithm fails, as it takes the best solution step by step and does not analyse the problem globally.

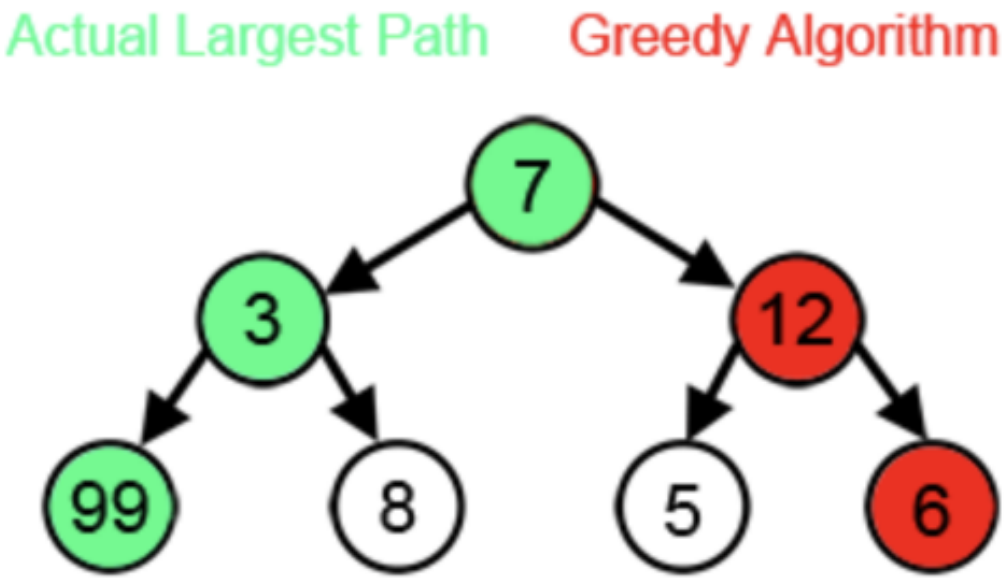


Figure 6. Visualization of a failure of the greedy algorithm in solving a problem (Bajaj, u.d.)

The greedy algorithm is often confused with dynamic programming. The first difference is that dynamic programming takes decisions at each stage based on the current problem and the solution to a previously solved subproblem for determining the globally optimal solution (GeeksforGeeks, 2021). Since the greedy algorithm finds locally optimal solutions, it is difficult to solve the same problem with it.

The advantages of the greedy approach are that it is fast and manageable to implement. In our program, we use it to find available time slots within a given interval of time. The algorithm solves the problem by inserting the tasks into the first available time slots in the calendar. Though, it is a primitive solution because the greedy algorithm can only do the simple insertion.

In the next section, we will describe the rules for our program that we have implemented in the code and why we need them.

Rules of the program

To schedule the tasks into a calendar that contains other events, we have considered some rules for splitting, placing, and prioritizing the tasks. These rules will help us to ensure that there is enough time to complete the tasks. In this section, we go through a set of rules we created upon the development process.

We have created a property for the task object which will handle the prioritization of the tasks. We use the Enum class to create a list of constants representing the priority levels LOW, MEDIUM, and HIGH. The tasks are sorted by the highest value. Moreover, we have considered adding another constant "exam" to point out the exam activities to ensure that the preparation for the exam will always be prioritized. We do not apply this rule in our program, though it is possible to expand the list of Enum constants in the Tasks.java file.

We also implement a rule that will avoid creating conflicting events in the calendar. E.g., if there is already a scheduled event between 11 and 13, a new event cannot overlap this period. The program will check if the calendar contains any events within the work time interval in the time range from the next day until the day of the task deadline, and then it places tasks into the calendar in the available slots. If the number of hours of one task is more than the number of free hours, the program splits the tasks into more events and places them in the available slots.

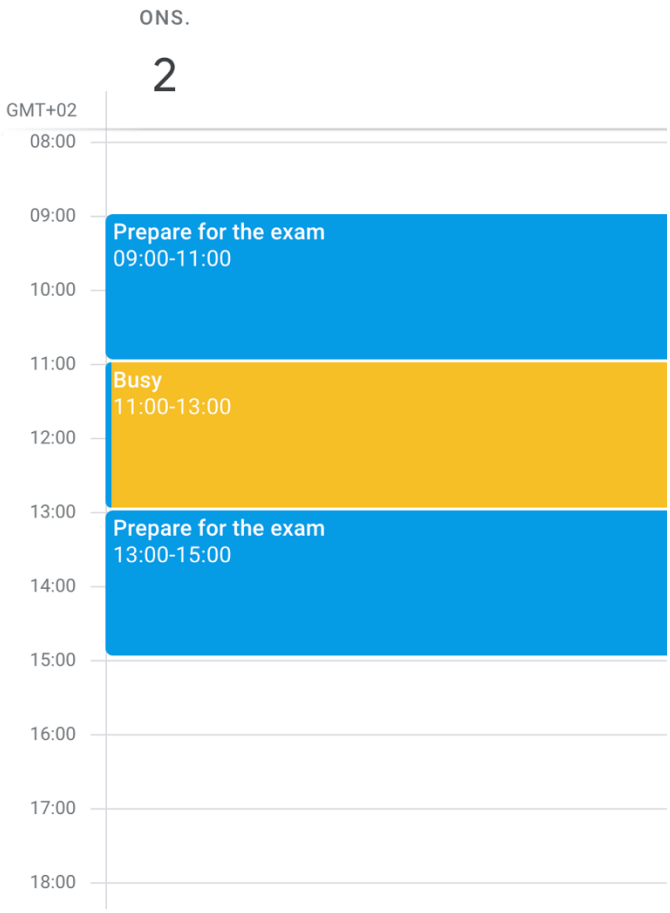


Figure 7. Task Manages splits an event in case of other events in the calendar

Code description

In this section, we will dive into the code and explain how we approach the problem. We will explain what an API is and how we use it in our project, along with the Java Collections interface and the JodaTime package, which has the necessary time and date classes. Furthermore, we will also explain the algorithms and implementation of the sorting methods in the code for organizing and dividing the tasks into sets of events.

Google Calendar API

Application programming interface (API) is a software interface that allows the interaction and data exchange between two applications independently from the programming language used to program them (MuleSoft, 2021).

In our project, we use the Google Calendar API. It makes the interaction between our program and Google Calendar possible through sending and receiving API calls (Google Inc., u.d.). Google Calendar can store private and shared calendars, as well as subscribed calendars. E.g., the latter can contain the schedule of the courses provided by universities. Using the Google Calendar API, we are no longer required to design and develop a calendar to use our program. Instead, the Task Manager sends the events into the personal Google Calendar of a user. Herewith, the user keeps the entire schedule in one platform, which is more convenient than shifting between many calendar platforms.

Google Calendar API supports various languages, such as Java, Python, PHP, and so on. To use this API, we had to select a programming language and create a project that will send and receive the API calls from the Google Calendar. Thus, our program is limited to work with only one calendar service and cannot be used with other calendar services, e.g., iCloud or Outlook.

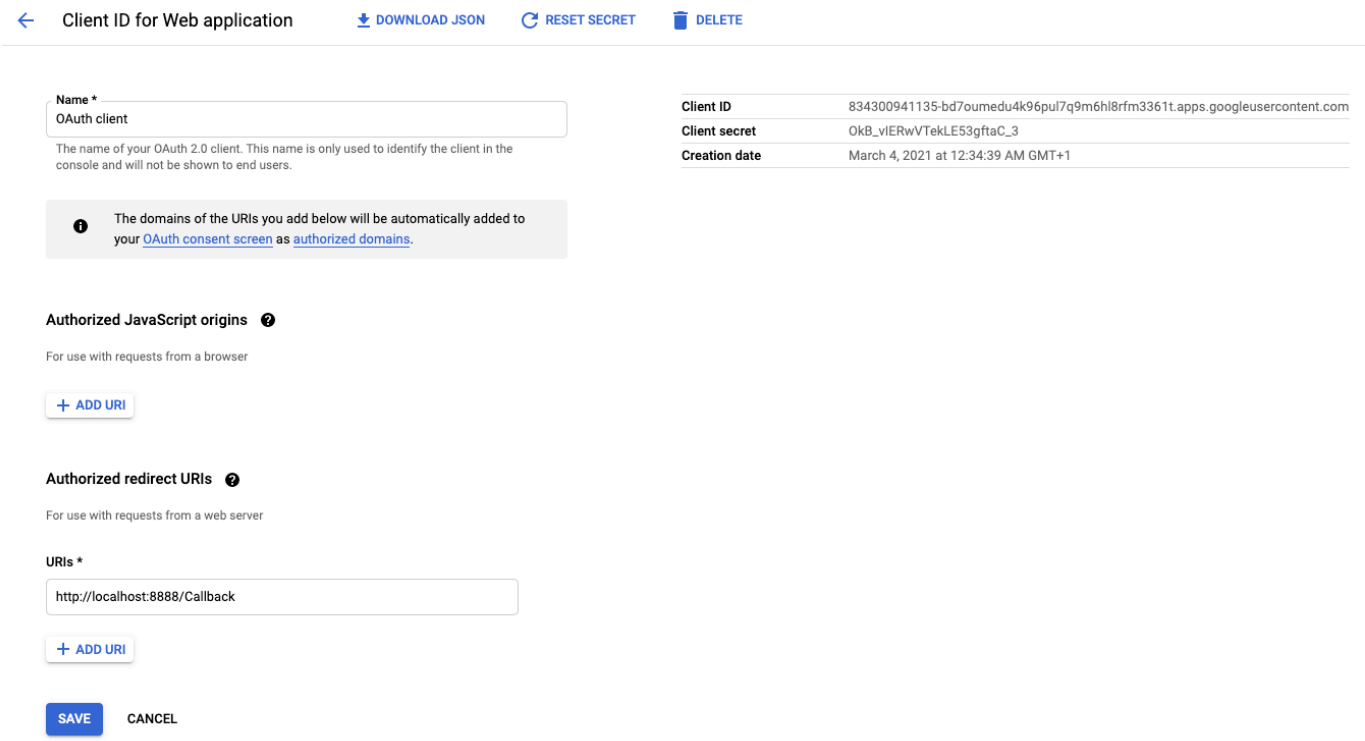


Figure 8. Google Cloud Platform (Google Cloud Platform, u.d.)

Another challenge is its safety requirements when connecting our program to a Google Account. It required many authentication approvals from Google, which has taken some time to resolve. We had to create a client ID for our application and add a .json file that contains the credentials for our Google Account. We have generated it at the Google Cloud Platform. It involved adding an authorized redirect URL http://localhost:8888/Callback which opened a Google Authentication page and asked for permission to use the Google Account together with our app.

The Google Calendar API is simple to implement and integrate into the code. It provides methods that can retrieve the information from the calendar, such as other events that are already scheduled. This helps us with finding the available slots for new tasks and sending events to the Google Calendar.

Java Collections

In our project, we use Java Collections, which gives us access to some pre-packaged data structures. It also includes methods to handle the data structures, such as the method `Collections.sort()` to organize the list of tasks that the program will receive from the user (edureka!, 2020). This makes the development process in Java easier. We also describe the use of LinkedList as a data structure and mergeSort as an algorithm for organizing the tasks in the following sections.

Collections framework

We use a Java Collection framework, which provides a List interface that extends the Collection interface. In our code, we make use of a class LinkedList for storing the tasks. LinkedList implements a List interface, which means that it applies the methods stored in the List interface (JavaTPoint, u.d.). The principal benefits of using LinkedList over the other provided classes in Java Collections are its dynamic size and ease of object insertion and deletion (Parvex, 2020). The picture below shows the architecture of the Java Collections, where it is shown that the LinkedList implements the List interface. This is called polymorphism.

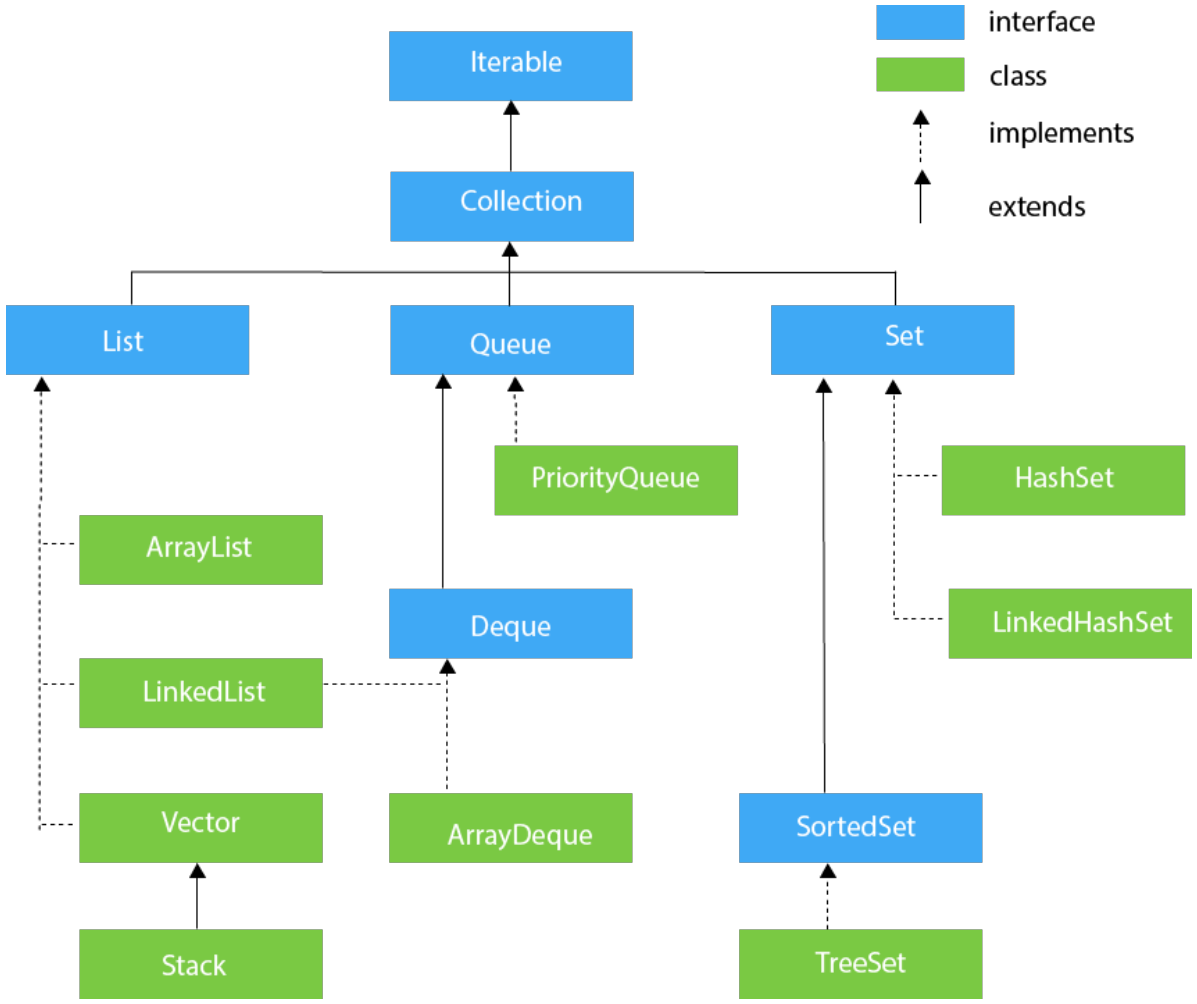


Figure 9. Hierarchy of Collections framework (JavaTPoint, u.d.)

Java Collection provides access to various methods such as `Collections.sort()`, which we use on LinkedList with tasks to arrange them by properties, such as deadline, priority, and duration. We will dive further into the method `Collections.sort()` in the section about Java Comparator.

LinkedList

As we have already mentioned, a Doubly LinkedList is one of the Java Collection classes. LinkedList is a linear data structure consisting of nodes, which represent a separate object (GeeksforGeeks, 2021). Every node in the Doubly LinkedList is linked to the next and previous nodes (Oracle, u.d.).

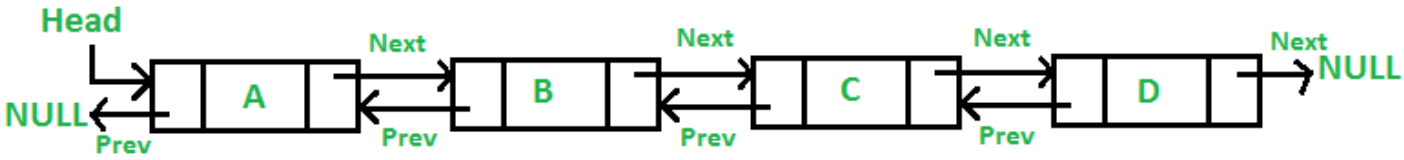


Figure 10. Visualization of a Doubly LinkedList with the nodes and data inside of them (GeeksforGeeks, 2021)

The nodes in LinkedList contain data and the information which links to the following node in the list. The first node in the LinkedList is called the head node (GeeksforGeeks, 2021). Doubly LinkedList has the advantage to insert elements at the front, end, after, or before a given node of the LinkedList (GeeksforGeeks, 2021).

In our project, we have created a Java class Tasks where we initialize the LinkedList for storing the tasks that the user creates. The user fills in the information about each task, such as the title, duration, priority, and deadline. The tasks are added to the LinkedList as Task objects with their properties. Afterwards, the user can request a list of all the tasks by choosing "Show all entered tasks" in the start menu. They are arranged using the `Collections.sort()` method provided by Java Collections. This method uses the mergeSort algorithm with the LinkedList data structure, and in the next section we will describe how the mergeSort works.

MergeSort

There are many different types of efficient sorting algorithms in Java, such as bubble sort, insertion sort, selection sort, and merge sort (Zobenica, 2020). The Java Collection framework implements the quickSort and mergeSort algorithms, among others. We use a LinkedList data structure that implements a List interface, and therefore the method `Collections.sort()` implements the mergeSort. However, the Collection sorting uses both mergeSort and quickSort for different data structures (GeeksforGeeks, 2020).

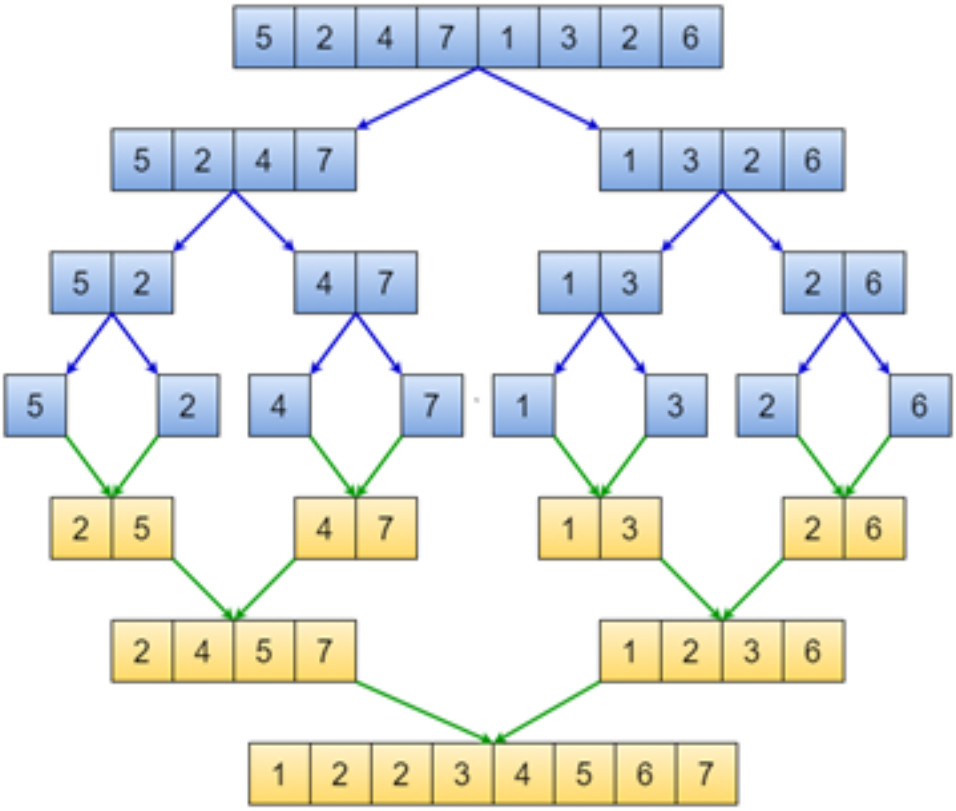


Figure 11. Visualization of the mergeSort algorithm (Gupta, 2020)

MergeSort uses a Divide-and-Conquer approach that splits a LinkedList in half until the last single element and then recursively merges the elements, sorted. Thus, the algorithm divides the list, then conquers the subproblems by solving them, and in the end, combines the solutions into a new sorted list (StudyTonight, u.d.).

As we do not limit the number of tasks that can be created at once, the advantage of mergeSort is that it can be applied to any size of data set. Though mergeSort shows efficiency decrease on smaller datasets, compared to other sorting algorithms, such as QuickSort. MergeSort also requires more space when executed as it needs to store the auxiliary arrays (GeeksforGeeks, 2021).

The time complexity for mergeSort

In our program, we use mergeSort to sort the elements in the list of tasks. The mergeSort algorithm is recursive, which means that it calls itself over and over. Therefore, the time complexity for such an algorithm will be O(NlogN) (GeeksforGeeks, 2021).

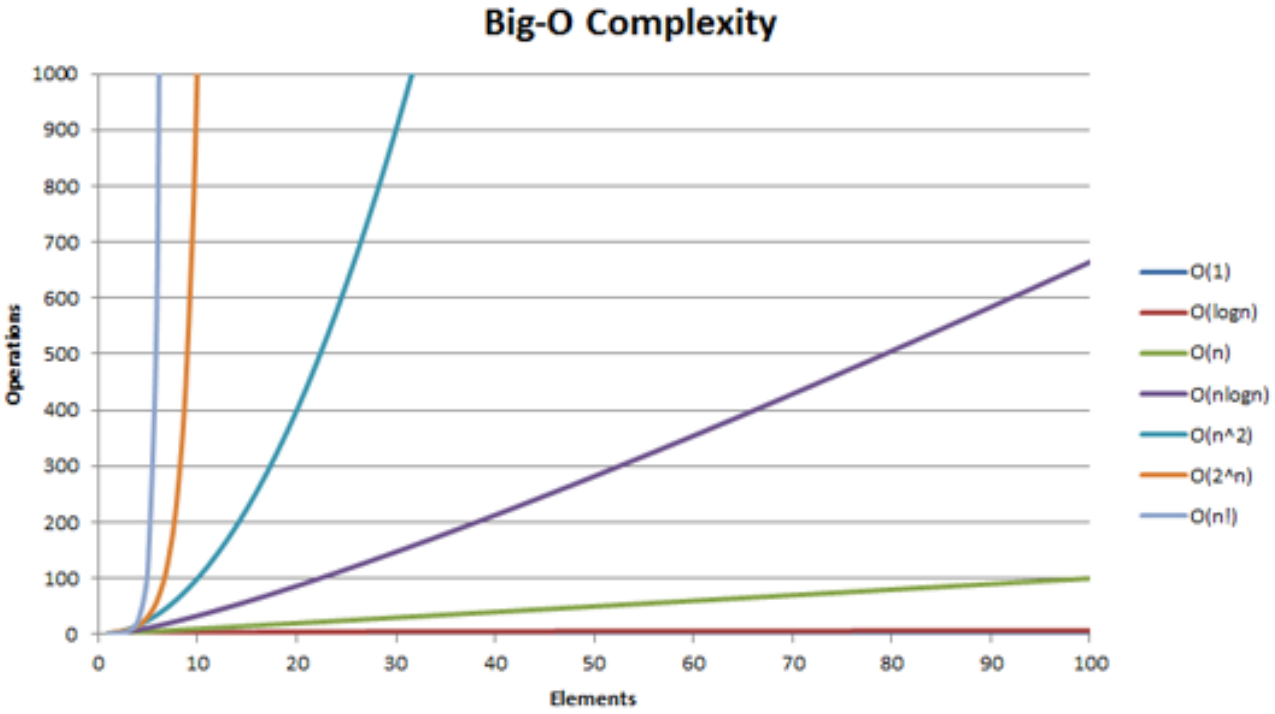


Figure 12. Time complexity for mergeSort is O(NlogN) (Webb, 2021)

Java Comparator

To create a sorting order, we created the properties the tasks should be sorted by. To create an efficient order of the tasks, we sort the tasks first by the deadline of each task to avoid placing the tasks with a distant deadline first. Then we sort by the priority of the tasks to ensure that the tasks with higher priority will move up in the list, though having the same deadlines. Finally, we sort by the duration, so the tasks with the lowest duration should come to the top of the list.

We use the method `Collections.sort()` from the Java Collections framework to arrange the list of user assignments. First, we define a Comparator where we give instruction on how and in which order the list should be sorted. Here we create 3 rules for sorting the tasks. The first element that gets sorted is the deadline of the tasks, where we apply the `compareTo()` method provided by the JodaTime package. The second element that should be sorted is the priority of the tasks. Here we use `compare()` method provided by the Integer class because we compare the numerical values of the provided priority Enum value. The third element to sort is the duration of the tasks where we apply the same `compareTo()` method from

JodaTime, as for the deadline. Herewith, we provide a sorting order for the `Collections.sort()` method.

```
// sort the tasks using the comparator
Comparator<Task> taskCom = new Comparator <Task>() {
    @Override
    public int compare(Task task1, Task task2) {
        int comparison = 0;
        comparison = task1.getDeadline().compareTo(task2.getDeadline());
        if (comparison == 0){
            comparison = Integer.compare(task1.getPriority(), task2.getPriority());
        } else if (comparison == 0){
            comparison = task1.getDuration().compareTo(task2.getDuration());
        }
        return comparison;
    }
};
Collections.sort(listOfTasks, taskCom);
```

Figure 13. Comparator in the Java class Tasks

We use the Collection sorting method by giving the set of data with the task objects together with the Comparator. One of the properties we use to arrange the tasks is the priority. We created this attribute as a constant variable of the Enum class. In the next section, we will explain the usage of the Enum class and how we implement it in the code.

Enum

Enum is a class that represents a group of constants. We use an enum to create a representation of a number (Baeldung, 2021). We use an enum class to determine the 3 levels of priority: HIGH, MEDIUM, and LOW, which are constant variables given a numerical value that corresponds to the priority level. Using the method `getValue()`, we get the numerical value of enum, which is used for organizing the tasks by their priority with Comparator. The enum class is included in our Tasks.java file.

```
// create a constant for priority values of type Enum
enum priorityEnum{
    LOW( value: 3),
    MEDIUM( value: 2),
    HIGH( value: 1);

    private final int priorityValue;
    priorityEnum(int value) { this.priorityValue = value; }
    public int getPriorityValue() { return this.priorityValue; }
}
```

Figure 14. Constant values

In the next section, we explain why we use the JodaTime classes to work with time and date instances. We will give examples of the methods we apply in the code and their advantages over the Java time package.

JodaTime

JodaTime is an open-source API that provides a quality alternative for java.time package with time and date classes and methods. JodaTime is user-friendly and comprehensive, as well as it is simple to implement into the code when working with time and date entities.

```
int daysCount = days.getDays();
```

Figure 15. The function `getDays()` provided by the JodaTime

JodaTime uses straightforward field accessors, such as *`getYear()`*, *`getDayOfWeek()`*, *`getDayOfMonth()`* etc. (Joda-Time, u.d.). JodaTime uses a Time Zone Database, also known as the TZ database, updated multiple times a year, and it integrates all changes made to this database (Joda-Time, u.d.).

We use the JodaTime package because it provides methods and functionalities that we need in our program. E.g., we use methods `plusDays()` to add days to a date and `plusMinutes()` to add minutes to the time in the day. We use them in the `handleTasksToTheCalendar()` method in the Tasks Java class.

```
// get the current day for looping through the events
DateTime currentDate = start.plusDays(u);
// get the end of the current day by adding 23 h and 59 m in minutes
DateTime endOfDay = currentDate.plusMinutes(1439);
```

Figure 16. Examples of using the JodaTime classes and methods

Moreover, the documentation states that the JodaTime methods are executed faster, implying improved performance, compared to the ones in java.time package (Joda-Time, u.d.).

In our code, we use the JodaTime provided DateTime, LocalTime and Interval classes. In the next section, we explain how we use the JodaTime Interval class in the code. We come with more examples of the use of JodaTime classes in the code in the chapter with class descriptions.

Intervals

We use intervals for determining the hours during the day when the user can solve scheduled tasks. We have created a function where a user can enter the start and end hours of the work time interval. We use the JodaTime class Interval with start and end entities, where we store the user-defined interval. We look for all the events in the calendar within the interval hours. Then we create an array of integers for storing all available time slots with start and end hours.

The first thing we do in the `handleTasksToCalendar()` method is using the for-each loop to find all events in the given calendar starting from the next day and until the deadline of the task. This helps us to calculate the available hours inside the given daily work time interval. Herewith, we get the available slots where the task events can be placed. The events are pushed to the calendar one by one, so the method repeats for each task. Once the task events are sent to the calendar, they fill in the available time slots. The process applies to each task in the list that is to be pushed to the calendar. The range of days that the Task Manager should loop through to find free time slots differs according to the task deadline.

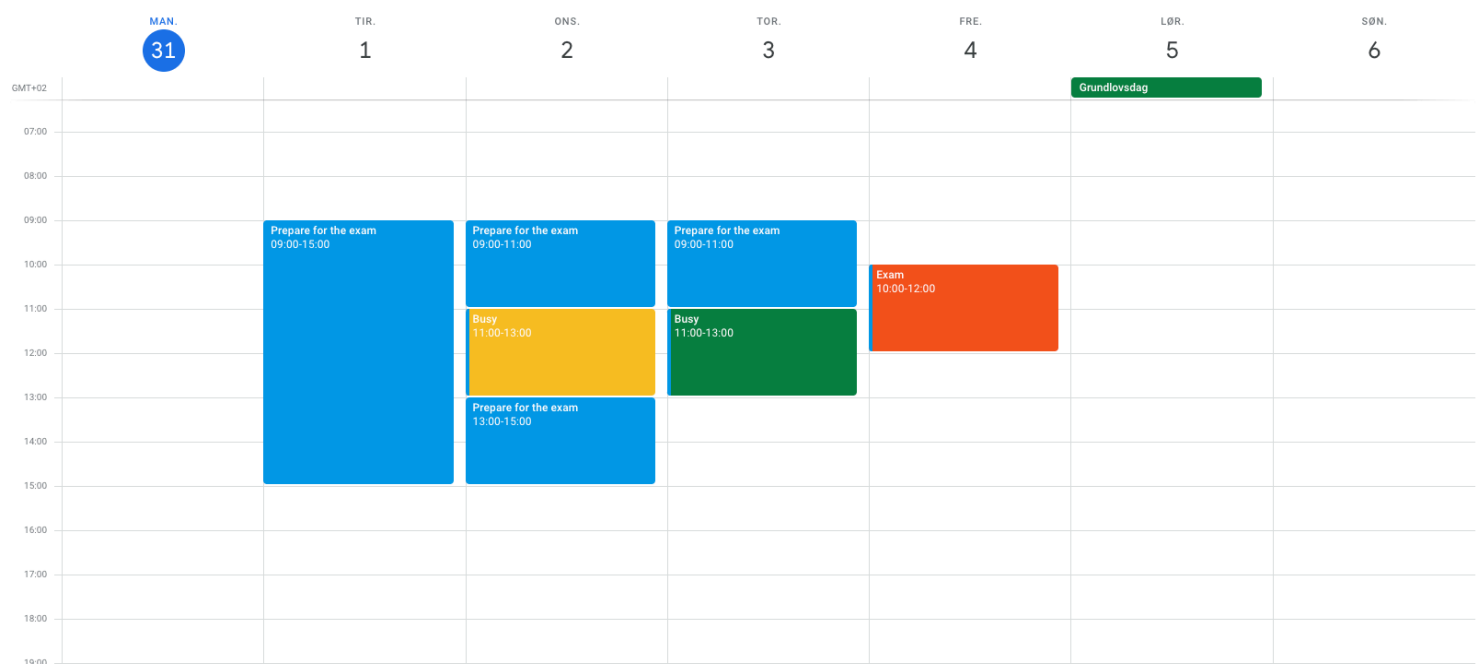


Figure 17. The events are inserted within the work time interval defined by the user

E.g., if a user adds a task of 10 hours duration, and the work hours are set from 9:00 to 15:00, it will be split into smaller events. The same applies if the free hours during the day interval are less than the task duration. Moreover, the number of hours extends to 12 by adding the buffer time of 2 hours. This implies subtracting the number of hours from task duration and placing them into the available time slot. Following this, it needs to look for other available time slots for the remaining task hours. If an event between 11 and 13 on the following day is found, another 2 hours will be placed between 9 and 11. The other 2 hours will be placed between 13 and 15. The remaining 2 hours will be placed on the following day between 9 and 11.

We will move on to the chapter where we will explain the usage of buffer time in the Task Manager.

Buffer time

Buffer time is an essential part of time management, which implies maximization of the time spent on a task. Buffer time can either be used or not, depending on whether extra time is needed. E.g., a meeting length can be estimated to be 2 hours, but if technical issues or a topic discussion occur, the meeting can take longer than expected. In this case, the pre-estimated duration can be expanded with the buffer

time, which will not cause inconvenience to the participants, as the buffer time is previously included for covering unexpected circumstances.

According to an article on Velaction, the buffer time is a part of project management and can be split into two types: buffer time for a project and buffer time for a task (Velaction, u.d.). The difference between buffer time for a project and a task is the way of calculating it.

Buffer time for a project calculates from the estimated work length and is placed in the last development phase of the project. So, if the project team cannot finish the project before the deadline, then the remaining tasks are exceeded to the buffer time.

On the other hand, the buffer time for a task gives a time slot for a person to finish a task that can be part of a larger project. Buffer time, in this case, is calculated according to the person's schedule, amount of work and deadlines. It is up to a person whether to use buffer time or not on working on other tasks that were not planned for the day (Velaction, u.d.).

In our project, we have chosen to add buffer time to a task proportionally to its duration, meaning adding the buffer time to the whole duration of the task. We implement a formula that expands the duration to 120%, which becomes the new duration, including the buffer time. The user can take advantage of the buffer for a break or completing the task. Herewith, the users can structure the given buffer time according to their needs.

Subconclusion

In this chapter we have listed the tools we use in making the structure of the program and storing and manipulating with different types of data.

In the next chapter we will dive into the source code and which solutions we have taken to create our application. One of the functions described here the buffer time calculation and its implementation.

Class description

In this section, we describe some of the Java classes we have created in the source code. We go through the code and explain the logic and the methods that we find essential for our program.

CalendarQuickstart

The Java class CalendarQuickstart contains the main method of the program, which executes when the program runs. As this is the main class, we have chosen to place a menu with the options to select. The `renderMenu()` method, which presents a menu with these 5 options:

```
public static void renderMenu(){
    System.out.println("\nScheduling Menu\n");
    System.out.println("1. Set you working time interval");
    System.out.println("2. Add new task");
    System.out.println("3. Send tasks to the calendar");
    System.out.println("4. Show all entered tasks");
    System.out.println("5. Exit");
}
```

Figure 18. Options in the Task Manager main menu

The scheduling menu consists of setting up the preferred work time interval, adding tasks and sending them to the Google Calendar. There is also an option to see a list of tasks. To choose any of these options, we use a switch as a function distributor. It contains 5 cases with corresponding method for execution.

```java
boolean exit = false;
String userChoice;
// create main menu for the user
while (!exit) {
    renderMenu();
    userChoice = scan.nextLine();

    switch (userChoice) {
        case "1":
            hours = timeInterval.setTimeInterval();
            break;
        case "2":
            tasks.addTask();
            break;
        case "3":
            //call a function to send the task to the calendar
            tasks.handleTasksToCalendar(hours);
            break;
        case "4":
            tasks.showListOfTasks();
            break;
        case "5":
            exit = true;
            System.out.println("See you!");
            break;
        default:
            System.out.println("Wrong choice, try again!");
            break;
    }
}
```

Figure 19. Switch used as a function distributor

In the `renderMenu()`, we added a default case, which will be executed if the user input does not correspond to any previous options. The user will be informed about the error, so the user will be sent back to the main menu select an action again. The switch, together with the `renderMenu()`, is wrapped into a while loop. The program should show the main menu, receive the user input, and execute the requiring function as long the program runs. We use a boolean datatype with a variable "exit" to mark whether the user continues using the program or wants to exit.

TimeInterval

```
do {
    try {
        System.out.println("Set a time interval when you can work during a day: ");
        boolean validStart = false;
        do {
            try {
                System.out.println("Set the start hour (HH): ");
                String startStr = scan.nextLine();
                this.start = TimeData.convertToHour(startStr);
                validStart = true;
            } catch (Exception e){
            }
        } while (!validStart);

        boolean validEnd = false;
        do {
            try {
                System.out.println("Set the end hour (HH): ");
                String endStr = scan.nextLine();
                this.end = TimeData.convertToHour(endStr);
                validEnd = true;
            }catch (Exception e){
            }
        } while (!validEnd);

        periodInterval = new Interval(this.start, this.end);
        dayInterval[0] = Integer.parseInt(getStart().toString( pattern: "HH"));
        dayInterval[1] = Integer.parseInt(getEnd().toString( pattern: "HH"));
        System.out.println("Your working time will start at " + dayInterval[0] + " and end at " + dayInterval[1]);
    } catch (Exception e){
        System.out.println("Wrong format! Please try again...");
    }
} while(periodInterval == null);
```

Figure 20. Piece of the method for setting the work time interval

As we have mentioned before, we use work time intervals to look for the free time slots and send them the events. The TimeInterval class contains the method `setTimeInterval()` for receiving the start and end hour of the day when a user wants to have the events placed. First, we create an Interval variable periodInterval, which we initialize as null. That means the periodInterval does not have the start and end hour yet. The method asks the user to insert these entities and then parses them from a String to a DateTime type. When the program receives valid data, it assigns it to the periodInterval.

Task and Tasks

We have created two separate Java files with classes Task and Tasks, where the Task class contains the constructor for the Task object, and the Tasks contains the methods to handle the task objects. The Task object has 4 properties - title, duration, priority, and deadline, which it receives from the user when creating a task.

```
private final String title;
private final int priority;
private final Duration duration;
private final DateTime deadline;

// create an object Task with five properties title, duration, priority and deadline
public Task (String title, Duration duration, int priority, DateTime deadline){
    this.title = title;
    this.duration = duration;
    this.deadline = deadline;
    this.priority = priority;
}
```

Figure 21. The constructor and properties of a Task object

All the properties of the Task are final variables, which means that they are constants and cannot be modified. We have made them as finals, as they should not change throughout the execution of the program.

As mentioned above, the Java class Tasks contains methods that handle the Task objects. Here we have created a method for adding tasks, `addTask()`, which asks for all the information needed to fill out the properties of the task object. In this method, we use the do-while loops when asking the user for several data. An example is a do-while loop for getting the task duration from the user input. We state first that the duration if equals to null, as the task has not received the duration data yet. The expression `while (duration == null)` indicates that the program needs to continue asking for the duration until the compatible data format is received. A try-catch statement helps, in this case, to ensure that the input format is acceptable. It also secures the program from crashing if the user has made a mistake in the input. If so, the program will display an error message. It will re-run the same

piece of code, asking for the duration once again. We give the user an unlimited number of tries, which means that the input tries are not tightened to a specific limit.

```
duration = null;
// ask for duration until the format is compatible
do {
    System.out.println("Set duration for your task (HH:MM): ");
    durationStr = scan.nextLine();
    String[] exploded = durationStr.split( regex: ":");
    try {
        int hours = Integer.parseInt(exploded[0]);
        int minutes = Integer.parseInt(exploded[1]);
        long hoursMillis = (long)hours * MILLIS_IN_HOUR;
        long minutesMillis = (long)minutes * MILLIS_IN_MIN;
        // new duration incl. buffer time of 20% of the task duration
        duration = new Duration((hoursMillis + minutesMillis)/100*120);
    } catch (Exception e){
        System.out.println("Something went wrong! Try again...");
    }
} while (duration == null);
```

Figure 22. Part of the code that allows the user to set the task duration

The program receives task length in hours and minutes converted into milliseconds to calculate and add the buffer time. We create a new Duration from JodaTime and use the sum of milliseconds to fulfil it.

```
deadline = null;
// ask for deadline until the format is compatible
do {
    try {
        System.out.println("Set deadline for your task (DD/MM/YYYY): ");
        deadlineStr = scan.nextLine();
        deadline = TimeData.convertToDate(deadlineStr);
    } catch (Exception e){
        System.out.println("Wrong date! Try again...");
    }
} while (deadline == null);
```

Figure 23. Part of the code that allows the user to set the task deadline

The JodaTime classes make it easier to work with date and time entities. E.g., when asking for the task deadline, the program requires a specific date format. Here we ask for a pattern "`dd/MM/yyyy`", which is received as a String. The specified pattern is used for converting the String into the DateTime class from the JodaTime library. For this purpose, we created a `convertToDate()` method, which takes the String of data and parses it to the DateTime format.

```
boolean isPriorityEnum = false;
int priorityValue = 0;
// ask for the priority until the format is compatible
do {
    try {
        System.out.println("Set priority HIGH, MEDIUM or LOW:");
        priority = scan.nextLine();
        priorityValue = priorityEnum.valueOf(priority.toUpperCase()).getPriorityValue();
        isPriorityEnum = true;
    } catch (Exception e){
        System.out.println("Something went wrong, try again!");
    }
} while (!isPriorityEnum);
```

Figure 24. Part of the code that allows the user to set the task priority

Another part of the code in the `addTask()` method is asking for the task priority. We start by giving the priority value of 0. As we have made this property an enum constant, we ask the user to enter a string containing one of the priority values. We have added the `toUpperCase()` method, so the user input turns to be case-insensitive. To change the value of the `priorityValue` variable, we request the numeric value that corresponds to the entered priority.

In the same class, we have developed a method that handles the tasks into the Google Calendar. The actions here include looping through the Google Calendar for creating intervals of free time slots, splitting the task by placing hours of its duration into the free time slots in the calendar.

As the program can receive many tasks, it needs to have a recursive method for looping through all the events and finding spots for them all in the calendar. Therefore, we start the function by placing it into a for-each loop, which will run for each task inside the LinkedList "`listOfTasks`". Here we use the try-catch statements as well to bypass crashing and throw an exception instead.

```
// foreach list of tasks and call the push to the calendar for each task
for (Iterator<Task> i = listOfTasks.iterator(); i.hasNext(); ) {
    // get intervals per day where the task can be placed and subtract the hours from the duration of the task
    try {
        // get all events from the calendar from tomorrow till the deadline of the task
        DateTime now = new DateTime();
        LocalDate today = now.toLocalDate();
        LocalDate tomorrow = today.plusDays(1);
        DateTime start = tomorrow.toDateTimeAtStartOfDay(now.getZone());
        Days days = Days.daysBetween(start.toLocalDate(), this.deadline.toLocalDate());
        int daysCount = days.getDays();
```

Figure 25. Creating the start and end day to find the free time slots using a for-loop

The program needs to get the work time interval to execute the method. The time interval is received from the user in the first option of the main menu. The tasks can be placed in the calendar starting from tomorrow and until the task deadline. Therefore, we use the LocalDate class to receive today and create the variable tomorrow by adding one day with the JodaTime method `plusDays()`. We also use JodaTime class Days to find the number of days between tomorrow and the task deadline to loop throughout that specific number of days looking for free hours.

```
// loop through the number of days until the day of deadline
for (int u = 0; u <= daysCount; u++){
    // get the current day for looping through the events
    DateTime currentDate = start.plusDays(u);
    // get the end of the current day by adding 23 h and 59 m in minutes
    DateTime endOfDay = currentDate.plusMinutes(1439);
    // start of the work time interval in the current day - user defined
    int currentHour = hours[0];
    // create an array of 2 positions for the available slots in the calendar
    // inside of work time interval with start and end hour
    int[] currentFreeHours = new int[2];
    // initialize the start of available time interval
    // with the hour of work time interval, and the end hour is not yet found
    currentFreeHours[0] = currentHour;
    currentFreeHours[1] = -1;
```

Figure 26. For-loop used to find free time slots in the calendar

We create a for-loop that receives the exact number of days it should loop through hour by hour to find the free time slots. We created an array to store all the intervals

of free hours. If there are no events during the day, we can use all hours in the work time interval to place the event. If any event is found, the program will loop through the hours. When the event is found, its start hour will be the end hour of the free time slot. So, the end hour index in `currentFreeHours[1]` will increase. As we already have the first free time interval, the start hour index in `currentFreeHours[0]` should change to `currentHour`.

```
// loop over the free hours and push the event to those hours
for (Iterator<int[]> a = freeHours.iterator(); a.hasNext(); ) {
    int[] currentArray = a.next();
    firstHour = currentArray[0];
    lastHour = -1;
    for (int e = currentArray[0]; e < currentArray[1]; e++) {
        taskHoursInMinutes -= 60;
        lastHour = e + 1;
        if (taskHoursInMinutes <= 0) {
            break;
        }
    }

    // create calendar object with the start hour and end hour
    com.google.api.client.util.DateTime startDate = new com.google.api.client.util.DateTime(currentDate.plusHours(firstHour).getMillis());
    com.google.api.client.util.DateTime endDate = new com.google.api.client.util.DateTime(currentDate.plusHours(lastHour).getMillis());
    String taskTitle = currentTask.getTitle();
    // push the even to the calendar
    this.googleCalendar.sendEventToCalendar(startDate, endDate, taskTitle);

    if (taskHoursInMinutes <= 0) {
        break;
    }
}
if (taskHoursInMinutes <= 0) {
    break;
}
```

Figure 27. Part of the function for sending the events to the calendar

When all free time slots between tomorrow and the deadline are found, the task can be split into events and sent to the calendar. At this step we also need to convert the JodaTime DateTime into the DateTime used by Google. We send the events with the title, start and end time to the Google Calendar.

Calculation of Buffer Time

In our code, we perform the buffer time calculation after receiving task duration from the user input. In the following code, we recalculate the task length finding the total amount of milliseconds in the task length and adding 20%. We use the formula in the 3rd line of code in the following screenshot:

```
long hoursMillis = (long)hours * MILLIS_IN_HOUR;
long minutesMillis = (long)minutes * MILLIS_IN_MIN;
// new duration incl. buffer time of 20% of the task duration
duration = new Duration((hoursMillis + minutesMillis)/100*120);
```

Figure 28. Part of the code where the buffer time is calculated

As a result, when the user enters 10 hours and 00 minutes as task duration, the task is created with 12 hours and 00 minutes of length instead. It is a simple buffer time formula that will be useful for the users when making a task. We have also considered a more complex calculation, which would admit more aspects, such as workflow and daily schedule. We will discuss this formula later in the discussion part of the report.

TimeData

In our source code, we have created a file TimeData class to store the methods that handle the time and date instances. Here we have created some functions to handle the different types of date and time formats. They generate user-friendly text with the time and date variables shown when displaying the list with tasks. It is also complicated to use the DateTime type attributes without formatting them, e.g., to organize the tasks.

The methods in TimeData class determine how different strings should be formatted. First, in the `convertToDate()` method, we specify which date pattern should be used to format a String to the DateTime. Here we use the DateTimeFormatter from JodaTime. To ensure that sending a String with the wrong date pattern will not interrupt the program execution, we have implemented a try-catch statement in the `convertToDate()` method. The method will detect the error and print a message that informs the user that they have entered the date in the incorrect format. We use `convertToDate()` when receiving the task deadline from a user.

```java
// convert deadline input into date of type DateTime
public static DateTime convertToDate(String dateStr) throws ParseException {
    try {
        DateTimeFormatter formatterDate = DateTimeFormat.forPattern("dd/MM/yyyy");
        return formatterDate.parseDateTime(dateStr);
    }
    catch (Exception e){
        System.out.println("Wrong date type! Try again...");
        throw e;
    }
}
```

Figure 29. Method for parsing a String with date pattern to DateTime

We have also created the `convertToTime()` method, which is similar to `convertToDate()`. This method parses an input of String type of the pattern "HH", which stands for hours of a day. Both methods accomplish the same purpose - converting a String into a DateTime object - but parsed from different input patterns. The `convertToTime()` is useful when asking a user to create a work time interval, where we receive the start and end hour.

```java
// convert interval input into date of type DateTime
public static DateTime convertToHour(String hourStr) throws ParseException {
    try {
        DateTimeFormatter formatterTimeStr = DateTimeFormat.forPattern("HH");
        return formatterTimeStr.parseDateTime(hourStr);
    }
    catch (Exception e){
        System.out.println("Wrong time type! Try again...");
        throw e;
    }
}
```

Figure 30. Method for parsing a String with hour pattern to DateTime

Another method is the `convertToHoursAndDays()` which receives an input in the form of an integer. This method is used to convert the duration of a task for printing it out for the user. This means that the user can expand the task duration to more than 24 hours. This method `convertToHoursAndDays()` will handle the given

duration and extract days, hours, and minutes from a task length. E.g., if the user creates a 37 hours and 30 minutes duration, the method will extract the number of whole days, complete hours, and remaining minutes. The result will be one day, 13 hours and 30 minutes.

```
// takes input as minutes and converts it into days, hours and minutes
public String convertToHoursAndDays(int x) {
    //since both are ints, you get an int
    int days = x / 1440;
    // subtracting days in minute equivalent co continue converting the resting minutes into hours and minutes
    x -= days * 1440;
    int hours = x / 60;
    int minutes = x % 60;
    String printResult = "";
    if (days >= 1) printResult = printResult + days + " days ";
    if (hours >= 1) printResult = printResult + hours + " hours ";
    if (minutes >= 1) printResult = printResult + minutes + " minutes";
    return printResult;
}
```

Figure 31. Method for retrieving the whole days, hours, and minutes

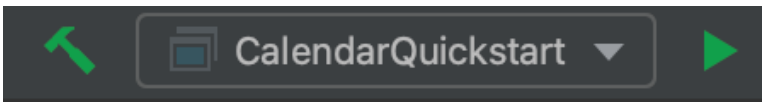
Subconclusion

In this chapter, we have gone through the essential parts of the program for scheduling the tasks. We have explained the purpose of each class, how we use the methods to perform the sorting and scheduling of the events.

We will now move to the user guide of our Task Manager application. Here we explain how to use our application, which options the user will be given, and show the application workflow.

User Guide

Start IntelliJ with the Time Management project and run it by pressing the play button in the upper right corner of IntelliJ window:



The program will start and display a menu with 5 options:

```
Scheduling Menu

1. Set you working time interval
2. Add new task
3. Send tasks to the calendar
4. Show all entered tasks
5. Exit
```

Type in a number corresponding to the option, which will execute the method behind it:

- The first option will request a start and end hour when a user wants the tasks to be scheduled. First, insert the start hour and then the end hour to generate a work time interval:

```
1
Set a time interval when you can work during a day:
Set the start hour (HH):
9
Set the end hour (HH):
15
Your working time will start at 9 and end at 15
```

  If the format is incorrect, an error message will be displayed, and a user will be asked to insert the hour again.

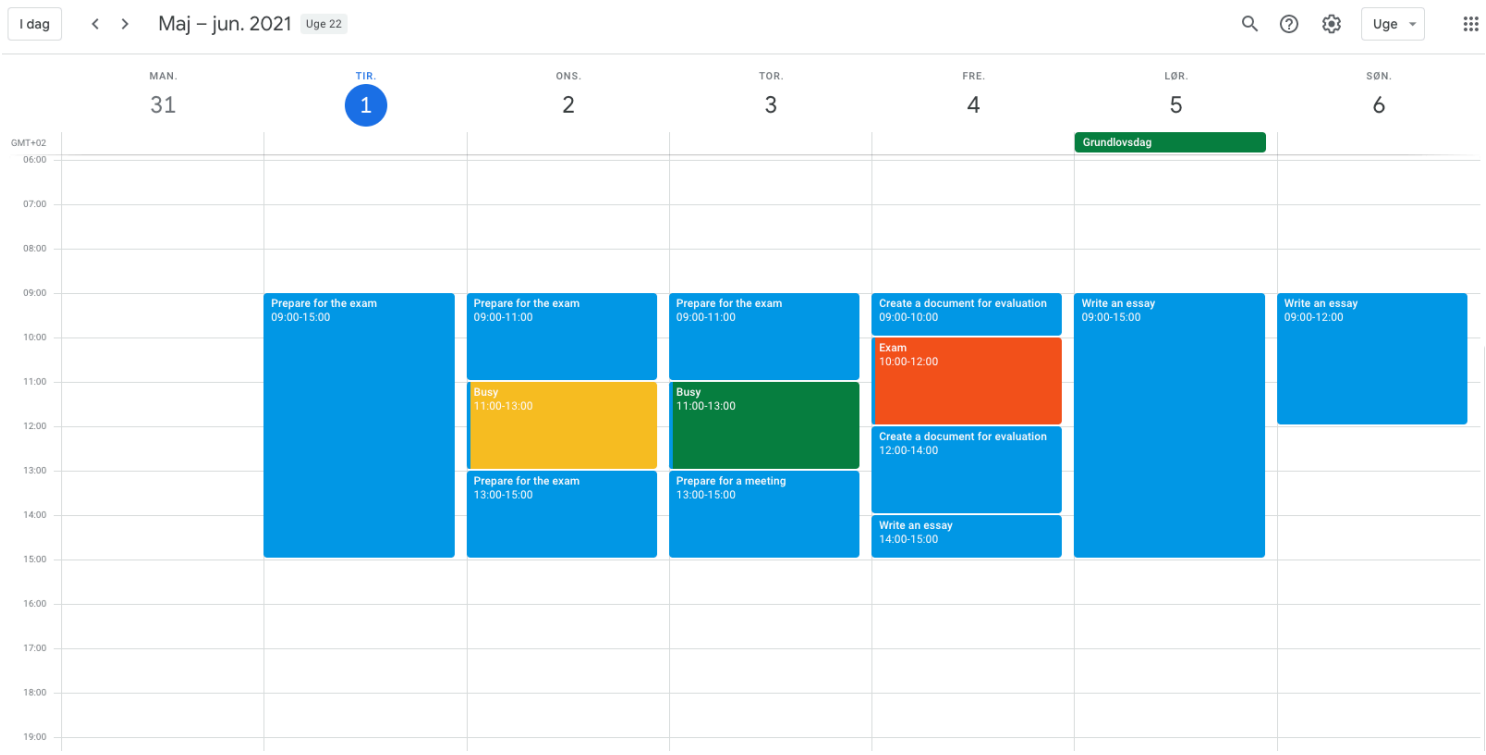
- The second option will start task creation, followed by adding them to a list and sorting it afterwards. The program will ask for information about the task,

such as title, duration, deadline, and priority level. The input pattern which the user should follow are also displayed when needed:

```
Create a new task!
Set title for your task:
Prepare for a meeting
Set duration for your task (HH:MM):
01:30
Set deadline for your task (DD/MM/YYYY):
04/06/2021
Set priority HIGH, MEDIUM or LOW:
High
Add more or exit?
exit
```

If more tasks should be added, the user can simply press ENTER, so the process will repeat.

- The third option in the menu is sending all the entered tasks to the Google Calendar.



- The fourth option executes the method which will trigger all the entered tasks and show them to the user. The previously sorted list of the tasks will be shown.

```
4

Task title: Prepare for a meeting
Task priority: 1
Task duration: 1 hours 48 minutes
Task deadline: 04/06/2021 00:00

Task title: Create a document for evaluation
Task priority: 2
Task duration: 2 hours 24 minutes
Task deadline: 08/06/2021 00:00

Task title: Write an essay
Task priority: 3
Task duration: 9 hours 36 minutes
Task deadline: 26/06/2021 00:00
```

If the tasks were already sent to the calendar, they will be removed from the list of tasks. Therefore, there will be displayed this message:

```
4
No tasks to show
```

- The fifth option on the menu will quit the program.

```
Scheduling Menu

1. Set you working time interval
2. Add new task
3. Send tasks to the calendar
4. Show all entered tasks
5. Exit
5
See you!
```

Viewing and editing the events after being sent to the calendar:

Go to your Google Calendar to see the events sent to the calendar from the Task Manager application. It is impossible to edit the events or move them around when they were sent to the Google Calendar. These actions should be done directly in the Google Calendar, where it is also possible to change the event title and duration. There the user can also drag and drop the events from one date to another.

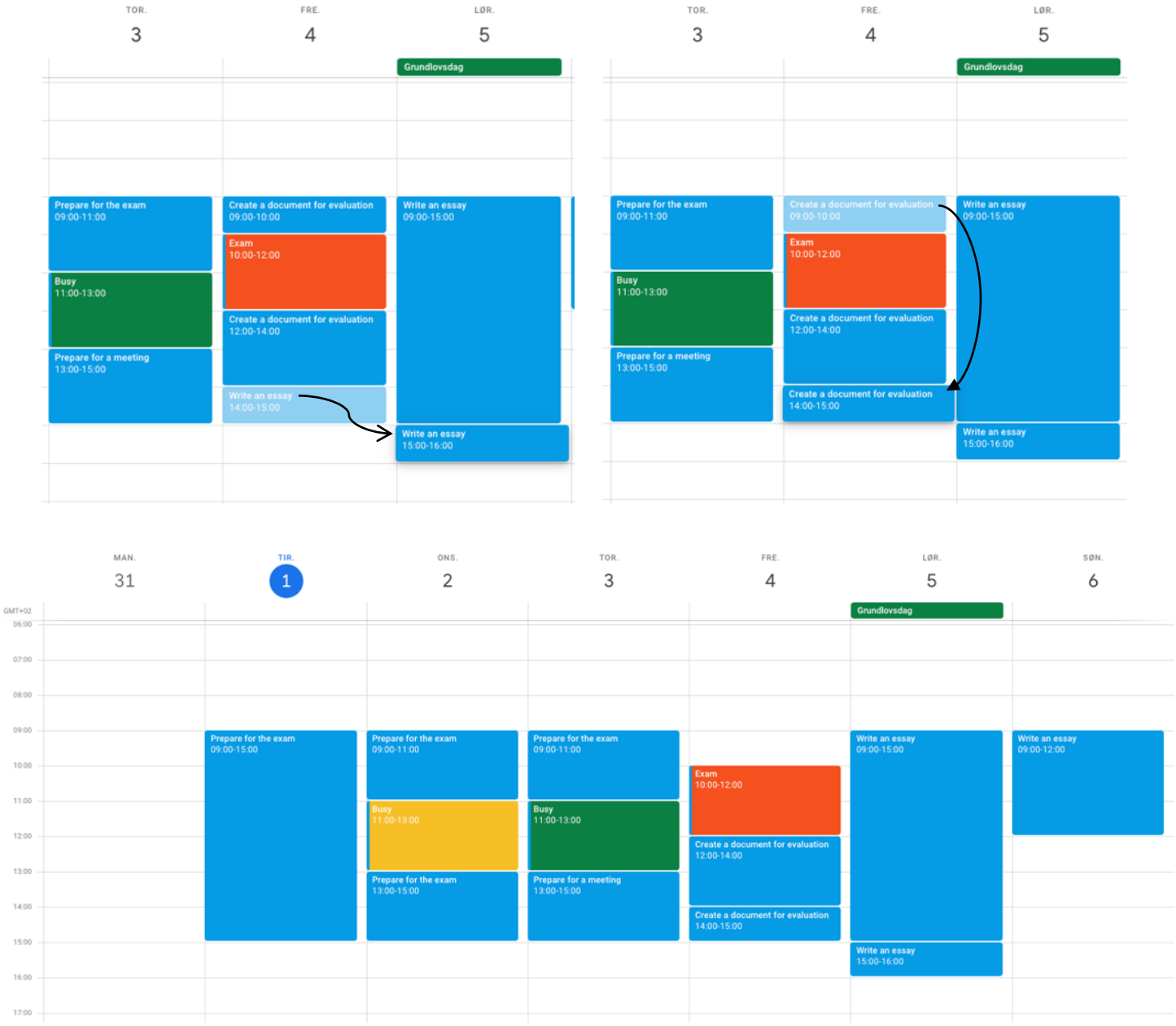


Figure 32. Drag-and-drop feature in the Google Calendar

Testing

During the development of our program, we tested the code frequently. Since our program consists of many classes and methods, we need to ensure that one element works before moving on to the next one. In this section, we will describe how we tested the program using the black box testing method.

Black box testing is a software testing method that checks the functionality of the program. It is completed without inspecting the structure of the source code. Black box testing focuses on the input and output of the software (Guru99, u.d.).

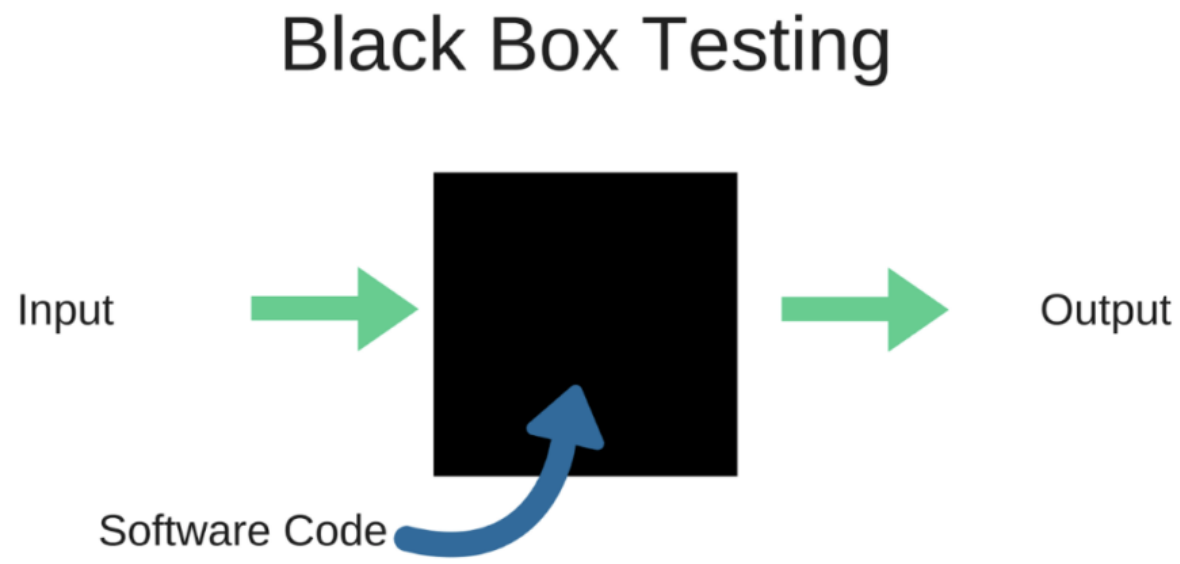


Figure 33. Black Box Testing technique (Testlio, u.d.)

The illustration above displays that the black box refers to no previous knowledge of the source code and that only the inputs and outputs are being analysed (Guru99, u.d.).

We have followed some steps for testing our application, as we have examined the positive test scenario, where we have checked if the program processes the valid inputs correctly. We had determined the expected outputs the program will give both to the valid and invalid inputs before running the test cases. E.g., we have tested if the valid date and time inputs are processed by the program as expected. We have also examined how the program will react when sending invalid inputs, i.e., the

negative scenario. At the end of the test, we have compared the actual outputs with the expected ones, fixed them and re-ran the tests.

```
1
Set a time interval when you can work during a day:
Set the start hour (HH):
9
Set the end hour (HH):
155
Wrong time type! Try again...
Set the end hour (HH):
15
Your working time will start at 9 and end at 15
```

Figure 34. Example of a negative test scenario

The picture above is an example of the tests we have run frequently during the development of the program. Here we test the program in the negative scenario by sending an invalid hour for the time interval. Since our input here parses into the DateTime variable, the hour cannot be outside the range of 00 to 23 hour.

Application interface

Our program has a command-line interface, which can be called inside of an IDE, e.g., IntelliJ, or executed through a Terminal window. This interface is simple but not user-friendly enough to keep the user. Therefore, it would be advantageous to expand the source code with the JavaFX libraries, which we would use to create a better interface for our application. Moreover, the Task Manager can be fully integrated with Google services by creating a browser extension.

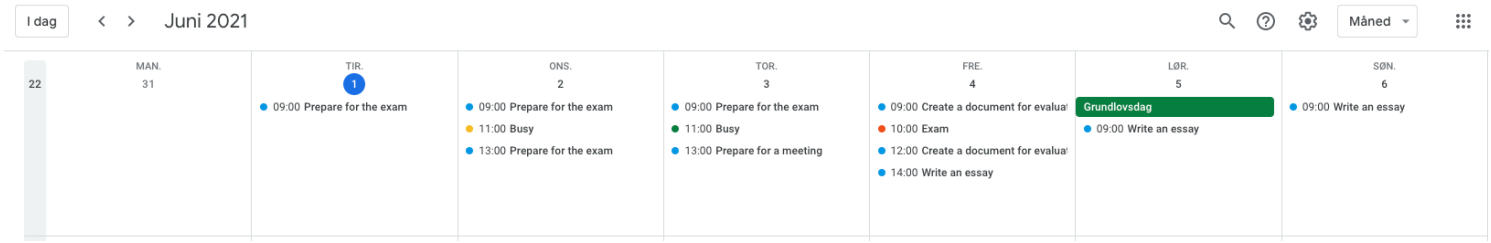


Figure 35. Google Calendar interface, monthly view

However, our program is also integrated with the Google Calendar through an API. The Google Calendar is another interface the user is using when the events are sent to the calendar. It is focused on using a separate calendar for the tasks sent from our app. The user can see the Google Calendar events when logged in to the personal Google Account.

Discussion

In this chapter, we raise some topics that we have worked with during the project preparation. We discuss how to optimize the formula for buffer time calculation and whether the one we implement is good enough.

Buffer time optimization

At the beginning of the project, we made a formula for finding the buffer time depending on the user day schedule. But after some consideration, we decided to simplify it by summing the original task duration with an additional 20% for the buffer. Though, the method we apply can result in several complications. Among them, adding 20% to a task with a long duration will result in a long buffer time. Indeed, the user can stop using the additional time. However, the task events do not show in the calendar how much buffer each event contains. On one side, the user can miscount the buffer time and end up with too much time remaining. On the other side, the user can end up lacking time. It can happen because of using the time irrationally and exceeding the buffer limit.

However, if the user spends less time completing the task, the created events for solving the same task will remain on Google Calendar. They will interfere with the Task Manager next time it loops through the calendar to find the free slots. It will not find free spots for multiple tasks because the completed tasks remain on the calendar. Herewith, the 20%-formula will not be efficient for planning events continuously. The expanded formula for buffer time calculation, taking into account the daily schedule, can resolve the issue.

We would calculate the buffer time out of the available time left in a day after subtracting all other events planned for this day. Using this formula, we could more

precisely find how much time the user may overrun and keep the schedule structured. Herewith, we assumed that considering the time spent on other daily activities would provide a more accurate buffer time.

To begin with, the user would have to provide a variety of factors that would assist in calculating buffer time. It includes providing minimum sleeping time, desired work time and breaks. Then, we would request all the scheduled during day activities from the Google Calendar and summarize their duration, which would give us the total duration of scheduled events during the day. Secondly, we would summarize it with data provided by the user. Then, all these factors subtracted from 24 hours gives the total available time left in a day. Here is our formula for this calculation:

> *Time left in a day = hours in a day – (scheduled events + total work hours + break time + minimum sleeping time)*

After finding out how much available time left, we would take the total work time for accomplishing the tasks and divide it by 24 hours. Afterwards, we would multiply the number we got with the available time left in the day. Finally, the calculation of buffer time would look like this:

> *Buffer time = (total task duration / hours per day) * time left in day*

Example:

Minimum sleeping time:	7 hours
Total work hours:	5 hours
Break time:	2 hours
Scheduled events:	4 hours
Time left:	6 hours

Task duration:	3 hours
Hours in a day:	24 hours
Buffer time:	45 minutes

Though, it is challenging to apply this formula as the user sets up a work time interval. The tasks split into events are sent to the calendar and fulfil the whole free space in the work time interval. Implementing the alternative formula will add the buffer time to each event already created for being placed in the free time slot. Herewith, we would then need to ignore the work schedule, which means moving the actual end of the workday to a later point in time. This formula helps plan the events without any defined work time interval making it easier to place in the calendar. Anyway, it requires some rules, i.e., to control not placing the events in the middle of the night.

Conclusion

A program for scheduling school assignments in the calendar can save the users' time and effort finding the compatible time to place the events. During the project development, we found out that making a task management application is a complex problem. Nevertheless, the tools we have chosen helped us develop a working program and answer the research question:

How can a scheduling algorithm be implemented, and how can we use the Google Calendar API to insert tasks generated in a program into a Google Calendar?

We have since the beginning created some delimitations, where one of them was using the Google Calendar API. It has provided a calendar interface, so we focused on the backend development. The use of API has made it simple to retrieve and send information to the calendar. Our class diagram helped us to create logic over the program. Here we have identified what methods each Java class needs and how they are related to each other.

Another challenge for creating a good scheduling program was finding the most optimal solutions. We have built up the logic of the program, which covers creating new tasks with the properties used for arranging the list of the tasks in the best possible order. We use each property of the assignment efficiently. E.g., we use the title to name an event in the Google Calendar, along with its duration and the task deadline. In the Comparator provided by the Java Collections, we use the duration and priority for instructing how to sort the assignments. We use the deadline and duration, together with the buffer time formula, to schedule the events into a specific period in the calendar, including the event splitting to avoid conflicts with other events. We have used the greedy approach for sending events to the first free spots. The `addTasks()` method, where we loop through the found free slots and push the events to those slots, represents the greedy approach. Furthermore, we have used the JodaTime package, which made it easier to code, but also it is simple to convert into the Google time classes.

The Task Manager is a program where we manage to integrate Google Calendar, so the Google users can benefit from planning the events and following the schedule using our program.

Bibliography

Git. (n.d.). *git*. Retrieved March 2021, from 1.1 Getting started - About Version Control: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Bajaj, V. (n.d.). Retrieved April 2021, from Greedy Algorithms: https://vikram-bajaj.gitbook.io/cs-gy-6033-i-design-and-analysis-of-algorithms-1/chapter1

MuleSoft. (2021). *MuleSoft*. Retrieved March 2021, from What is an API? (Application Programming Interface): https://www.mulesoft.com/resources/api/what-is-an-api

Google Inc. (n.d.). *Google Calendar API*. Retrieved February 2021, from https://developers.google.com/calendar

Google Cloud Platform. (n.d.). *Google Cloud Platform*. Retrieved March 2021, from Client ID for Web application: https://console.cloud.google.com/apis/credentials/oauthclient

edureka! (2020, September 15). Retrieved May 2021, from Java Collections - Interface, List, Queue, Sets in Java with examples: https://www.edureka.co/blog/java-collections/

JavaTPoint. (n.d.). *JavaTPoint*. Retrieved May 2021, from Collections in Java: https://www.javatpoint.com/collections-in-java

Parvex, F. (2020, July 20). *Great Learning*. Retrieved from https://www.mygreatlearning.com/blog/data-structures-using-java/?fbclid=IwAR1t6bzld-xDXqxuNBXiUfI4FL-pqHGXxvapLlWFmrqWfCENh_zsXqz7I2A#t2

Oracle. (n.d.). *Oracle Docs*. Retrieved May 2021, from Class LinkedList<E>: https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

GeeksforGeeks. (2021, April 13). *Greedy approach vs Dynamic programming*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/greedy-approach-vs-dynamic-programming/

GeeksforGeeks. (2021, March 5). *LinkedList in Java*. Retrieved from GeekforGeeks: https://www.geeksforgeeks.org/linked-list-in-java/

GeeksforGeeks. (2021, May 24). *Doubly LinkedList*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/doubly-linked-list/

GeeksforGeeks. (2021, April 18). *LinkedList Data Structure*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/data-structures/linked-list/

Zobenica, D. (2020, May 5). *Sorting Algorithms in Java*. Retrieved from StackAbuse: https://stackabuse.com/sorting-algorithms-in-java/

GeeksforGeeks. (2020, May 11). *Why Quick Sort preferred for Arrays and Merge Sort for Linked Lists?* Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/why-quick-sort-preferred-for-arrays-and-merge-sort-for-linked-lists/

Gupta, V. (2020, December 28). *Visualizing, Designing, and Analyzing the Merge Sort Algorithm*. Retrieved from gitconnected: https://levelup.gitconnected.com/visualizing-designing-and-analyzing-the-merge-sort-algorithm-cf17e3f0371f

StudyTonight. (n.d.). *Merge Sort Algorithm*. Retrieved May 2021, from StudyTonight: https://www.studytonight.com/data-structures/merge-sort

GeeksforGeeks. (2021, April 2021). *Quick Sort vs Merge Sort*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/quick-sort-vs-merge-sort/

Webb, C. (2021, May 11). *Complexity and Big-O Notation In Swift*. Retrieved from Medium: https://medium.com/journey-of-one-thousand-apps/complexity-and-big-o-notation-in-swift-478a67ba20e7

GeeksforGeeks. (2021, May 18). *Merge Sort*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/merge-sort/

Baeldung. (2021, May 20). *A Guide to Java Enums*. Retrieved from Baeldung: https://www.baeldung.com/a-guide-to-java-enums

Joda-Time. (n.d.). *Joda-Time*. Retrieved April 2021, from Joda-Time: https://www.joda.org/joda-time/

Velaction. (n.d.). *Buffer Time*. Retrieved April 2021, from Velaction: https://www.velaction.com/buffer-time/

Guru99. (n.d.). *Black Box Testing*. Retrieved May 2021, from Guru99: https://www.guru99.com/black-box-testing.html

Testlio. (n.d.). *Testlio*. Retrieved May 2021, from https://testlio.com/wp-content/uploads/2017/12/Screenshot-2017-12-06-16.51.30-1024x529.png