ParcelSorter-Smart Package Sorting and Routing Simulation Using Classical Data Structures

2- Introduction

ParcelSorter simulates a smart logistics center where parcels are received, categorized, and dispatched based on destination city, size, and priority. Inspired by real-world parcel processing, this project aims to model the system using fundamental data structures within a simulation environment. Parcels are categorized according to their destination, size, and priority, then processed and dispatched in an appropriate order. The simulation engine integrates five core data structures—queues, stacks, binary search trees (BSTs), hash tables, and circular linked lists—to build a dynamic and testable system.

3- Program Interface

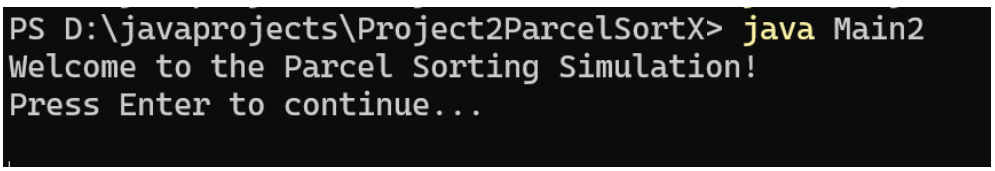
The program uses a file named **config.txt**, allowing the user to modify parameters and control the program's behavior. During runtime, no further input is required from the user. When the user starts the program, a welcome message is displayed in the terminal first, and the program waits for the user to press Enter to continue.

If the user wants to run the program using the command prompt (CMD), they must first navigate to the project directory, then compile the source files, and finally run the program using the Main2 class.

4- Program Execution

4.1) Opening

The user navigates to the directory, compiles the program, and runs it. The following output appears on the screen.



4.2) Config.txt FILE

The user can modify parameters in this file. The parameters that can be changed are as follows:

1. Hash table size
2. Size of the queue for arrived parcels
3. Number of ticks to be executed
4. Minimum number of parcels generated per tick
5. Maximum number of parcels generated per tick
6. Return probability of parcels
7. Cities
8. City queue size
9. How often the return stack will be emptied (in ticks)

4.3) Program Flow

4.3.1) Generating the parcels

Since this is a simulation, parcels are generated randomly. Each parcel is assigned an ID, destination city, priority, size, and the tick number when it was generated. Then, when the queue for parcels is not full, the parcels are added to the queue and their status is set to "in queue." When the queue is full, the parcels are placed in a *waiting stack*. After that, the size of the queue is displayed, the contents of the queue are shown, and the parcel tracker table containing the parcels is printed on the screen. For example:

```
--------------------------
New Parcel: ID=31
Destination City: Izmir
Priority: 2
Size: Small
Arrival Tick: 10

Enqueued parcel with ID: 31

--------------------------
New Parcel: ID=32
Destination City: Istanbul
Priority: 1
Size: Medium
Arrival Tick: 10

Enqueued parcel with ID: 32
```

```
Number of elements in ArrivalBuffer: 5
All parcels in Queue: 29 30 31 32 33

Parcel Tracker Table:
Index 0: [30 : InQueue] -> [20 : Sorted] -> [10 : Sorted] -> null
Index 1: [31 : InQueue] -> [21 : Sorted] -> null
Index 2: [32 : InQueue] -> null
Index 3: [33 : InQueue] -> null
Index 4: empty
Index 5: empty
Index 6: [26 : Sorted] -> [16 : Sorted] -> [6 : Sorted] -> null
Index 7: [27 : Sorted] -> null
Index 8: [28 : Sorted] -> [18 : Sorted] -> null
Index 9: [29 : InQueue] -> [19 : Sorted] -> null
```

Certain properties are checked at each tick. For opening the first tick always starts by generating parcels. Like I mentioned at *4.3.1*.

As for the elements being checked step by step.

4.3.2) The if block for queue of returned parcels

This queue is used when a returned parcel re-enters the cycle for re-sorting, but the queue of its destination city is full. In such cases, the parcel is transferred to this return queue. At the beginning of the if block, it is first checked whether this queue is full, and operations are carried out accordingly. If the queue is full, an informative message is displayed and inner loops starts. Before of that the contents of the queue are printed.

Then inner loop begins to try to empty the return queue. The process starts by dequeuing the first parcel. If the destination city's queue is still full, the parcel is re-added to the return queue. If the queue is not full, the parcel is sorted and its status is updated to "sorted." An informative message is displayed afterward.

```
Tick #16************************************************************
WaitingQueueforReturns is not empty, transferring parcels to sorter...
All parcels in Queue: 12 19 27 24

City queue is full, 12 will enqueue back.
Enqueued parcel with ID: 12

City queue is full, 19 will enqueue back.
Enqueued parcel with ID: 19

Parcel ID 27 is transferred from WaitingQueueforReturns to sorter.
Enqueued parcel with ID: 27


Parcel ID 24 is transferred from WaitingQueueforReturns to sorter.
Enqueued parcel with ID: 24
```

4.3.3) The if block for Return Stack

This loop is the block that enables parcels added to the Return Stack to be attempted for re-sorting. For this block to execute, the tick count must first be checked. The user can change this frequency via the config.txt file. After the tick check, it first verifies whether the stack is not empty. Informative messages are displayed for both cases. If the stack is not empty, the process continues.

Then, the stack is processed. Parcels are sequentially removed from the stack. Before attempting to sort each parcel, it is checked whether the queue of the parcel's destination city is full. If it is full, the parcel is moved back to the waiting queue for returned parcels, as mentioned above. If the city's queue is not full, the parcel is sorted.

```
Tick is multiple of 5, Transferring from Stack to DestinationSorter...

From Stack Parcel ID: 11
New status is 'Sorted'.
Enqueued parcel with ID: 11

From Stack Parcel ID: 5
New status is 'Sorted'.
Enqueued parcel with ID: 5

From Stack Parcel ID: 30
New status is 'Sorted'.
Enqueued parcel with ID: 30

City queue is full, will transfer to WaitingQueueforReturns.
Enqueued parcel with ID: 25
```

4.3.4) If block for parcels redirected to the waiting stack because the initial receiving queue is full

First, the process begins by checking whether the waiting stack (*mentioned in 4.3.1*) is not empty. An informative message is displayed, followed by printing the contents of the stack.

Then, a loop is entered to process the parcels in the stack. This loop continues as long as the initial receiving queue is not full and the waiting stack is not empty. Within the loop, parcels are removed from the stack and transferred to the initial

receiving queue, with their status updated to "inqueue". Then the initial receiving queue printed.

If the loop to process parcels from the stack is not entered, parcel generation as described in section *4.3.1* must be performed.

```
BufferStack is not empty, transferring parcels to ArrivalBuffer...
Parcels in Buffer Stack:
Parcel ID: 48 | Status: Waiting
Parcel ID: 47 | Status: Waiting
Parcel ID: 46 | Status: Waiting
Parcel ID: 45 | Status: Waiting
Parcel ID: 44 | Status: Waiting
Parcel ID: 43 | Status: Waiting
```

```
Enqueued parcel with ID: 48

Parcel id=48 | Status= InQueue
Parcel with ID 48 is transferred from BufferStack to ArrivalBuffer.
```

```
Enqueued parcel with ID: 47

Parcel id=47 | Status= InQueue
Parcel with ID 47 is transferred from BufferStack to ArrivalBuffer.
```

```
Enqueued parcel with ID: 46

Parcel id=46 | Status= InQueue
Parcel with ID 46 is transferred from BufferStack to ArrivalBuffer.
```

These were the required checks for each tick.

4.3.5) Creation of cities and adding parcels to city queues

The process begins with an informative header. Then, a loop is entered to sort the parcels. First, a parcel is dequeued from the initial queue (arrivalBuffer) For this parcel, there are two possible scenarios:

- If the parcel’s destination city is not yet in the BST, or if the city's parcel queue is **not full**, the parcel is added to the corresponding queue of that city in the BST using a specific method. Its status is then updated to **"sorted"** after displaying an informative message.

- If neither of these conditions is met, it means the parcel queue of the destination city is full. In that case, the parcel is added back to the initial queue from which it was dequeued, and its status remains **"inqueue."**

```
2

City queue is full couldn't sort, parcel will be enqueued back to ArrivalBuffer.
Enqueued parcel with ID: 20

2

City queue is full couldn't sort, parcel will be enqueued back to ArrivalBuffer.
Enqueued parcel with ID: 30

1

Parcel with ID 33 is being processed.
Enqueued parcel with ID: 33

2

City queue is full couldn't sort, parcel will be enqueued back to ArrivalBuffer.
Enqueued parcel with ID: 34
```

After this part, the sorted list of cities along with their parcels, the total number of cities, and the city with the most parcels are printed. Then updated table of parcels is printed.

```
Ankara
All parcels in Queue: 3 5
Antalya
All parcels in Queue: 4 6 8
Bursa
All parcels in Queue: 7
Izmir
All parcels in Queue: 2


The number of cities is 4

City with the most parcels: Ankara

Index 0: empty
Index 1: empty
Index 2: [2 : Sorted] -> null
Index 3: [3 : Sorted] -> null
Index 4: [4 : Sorted] -> null
Index 5: [5 : Sorted] -> null
Index 6: [6 : Sorted] -> null
Index 7: [7 : Sorted] -> null
Index 8: [8 : Sorted] -> null
Index 9: empty
```

4.3.6) Dispatching the parcels

In this part, the potential order of city activity is printed first. Then, to select the active terminal, a loop iterates through the list in order until it finds a city that has at least one parcel in its queue. Once the active city is selected, its parcels are retrieved, and another loop begins. In this loop, parcels are dequeued one by one from the city's parcel queue. Based on the probabilities defined by the user in the config.txt file, each parcel's status is updated to either **"dispatched"** or **"returned."**

If a parcel is marked as **"returned,"** it is also pushed onto the **Return Stack** (*mentioned in section 4.3.3*). After this, the contents of the Return Stack are printed.

Then, to simulate the possibility of the terminal remaining the same, the system proceeds to the city it is connected to once. Finally, parcels that were dispatched are removed and parcel tracking table is printed

```
Ankara
All parcels in Queue: 2
Antalya
All parcels in Queue: 5
Istanbul
All parcels in Queue: 3 6 7
Izmir
Queue is empty.


The number of cities is 4

City with the most parcels: Istanbul

Index 0: empty
Index 1: empty
Index 2: [2 : Sorted] -> null
Index 3: [3 : Sorted] -> null
Index 4: empty
Index 5: [5 : Sorted] -> null
Index 6: [6 : Sorted] -> null
Index 7: [7 : Sorted] -> null
Index 8: empty
Index 9: empty


Izmir -> Ankara -> Bursa -> Istanbul -> Antalya ->

Active City: Ankara
```

```
Dispatched parcels are removed from ParcelTracker table.

Index 0: empty
Index 1: empty
Index 2: empty
Index 3: [3 : Sorted] -> null
Index 4: empty
Index 5: [5 : Sorted] -> null
Index 6: [6 : Sorted] -> null
Index 7: [7 : Sorted] -> null
Index 8: empty
Index 9: empty
```

4.4) Ending

When the specified number of ticks is reached, the loop that tracks the ticks is exited. Then, some final summary information is printed to the screen:

- City with the most parcels
- Number of parcels each city had
- Total number of returned parcels
- Total number of generated parcels

```
City with the most parcels: Ankara

City Ankara had 12 different parcels.
City Antalya had 7 different parcels.
City Bursa had 4 different parcels.
City Istanbul had 9 different parcels.
City Izmir had 12 different parcels.
Total parcels returned: 14
Total generated parcels: 47
```

5) Reporting and Logging

In various parts of the code, logging is performed to two different txt files. The .txt file is cleared at the beginning of each application run. One of them is *DispatchedReturnedsAndReport.txt*. This file logs the status and ID of each parcel that is either dispatched or returned, along with various statistics about the total number of parcels. At the final tick, it also logs the number of parcels in

the parcel tracking table, categorized by their status. This file is more suitable for tracking parcel counts.

```
[2025-06-14 21:05:51] [INFO]: Parcel ID: 40| Status: Dispatched
[2025-06-14 21:05:51] [INFO]: Parcel ID: 26| Status: Returned
[2025-06-14 21:05:51] [INFO]: Parcel ID: 25| Status: Dispatched
[2025-06-14 21:05:51] [INFO]: Parcel ID: 34| Status: Returned
[2025-06-14 21:05:51] [INFO]: Parcel ID: 38| Status: Dispatched
[2025-06-14 21:05:51] [INFO]: Parcel ID: 21| Status: Dispatched
[2025-06-14 21:05:51] [INFO]: 47 parcels are generated.
[2025-06-14 21:05:51] [INFO]: 16 ticks executed.
[2025-06-14 21:05:51] [INFO]: 14 parcels are returned.
[2025-06-14 21:05:51] [INFO]: 32 parcels are dispatched.
[2025-06-14 21:05:51] [INFO]: Izmir is the city with the most parcels.
[2025-06-14 21:05:51] [INFO]: 5 is the maximum size of ArrivalBuffer reached during the simulation.
[2025-06-14 21:05:51] [INFO]: 7 is the maximum size of ReturnStack reached during the simulation.
[2025-06-14 21:05:51] [INFO]: 2 is the height of the Binary Search Tree (BST) used for sorting parcels by city.
[2025-06-14 21:05:51] [INFO]: 1.5 is the load factor of the ParcelTracker hash table.
[2025-06-14 21:05:51] [INFO]: Waiting = 0, InQueue = 3, Sorted = 10, Returned = 2
[2025-06-14 21:05:51] City: Ankara, Total Parcels: 12
[2025-06-14 21:05:51] City: Antalya, Total Parcels: 7
[2025-06-14 21:05:51] City: Bursa, Total Parcels: 4
[2025-06-14 21:05:51] City: Istanbul, Total Parcels: 9
[2025-06-14 21:05:51] City: Izmir, Total Parcels: 12
```

For *simulation_log.txt*, this file receives logs from various parts of the code. Below are some example outputs:

```
[2025-06-14 21:05:51] [INFO]: **********Tick #14 started.**********
[2025-06-14 21:05:51] [INFO]:
New parcel: ID= 38
Destination City= Antalya
Priority= 1
Size= Large
[2025-06-14 21:05:51] [INFO]: Enqueued parcel with ID: 38
[2025-06-14 21:05:51] [INFO]:
New parcel: ID= 39
Destination City= Izmir
Priority= 1
Size= Small
[2025-06-14 21:05:51] [INFO]: Enqueued parcel with ID: 39
[2025-06-14 21:05:51] [INFO]:
New parcel: ID= 40
Destination City= Istanbul
Priority= 1
Size= Large
[2025-06-14 21:05:51] [INFO]: Enqueued parcel with ID: 40
[2025-06-14 21:05:51] [INFO]: 3 parcels are generated, enqueued to the ArrivalBuffer and stated as InQueue.
[2025-06-14 21:05:51] [INFO]: Number of elements in ArrivalBuffer: 3
[2025-06-14 21:05:51] [INFO]: Parcel with ID 38 is dequeued from ArrivalBuffer.
[2025-06-14 21:05:51] [INFO]: Enqueued parcel with ID: 38
[2025-06-14 21:05:51] [INFO]: Parcel with ID 39 is dequeued from ArrivalBuffer.
[2025-06-14 21:05:51] [INFO]: Enqueued parcel with ID: 39
[2025-06-14 21:05:51] [INFO]: Parcel with ID 40 is dequeued from ArrivalBuffer.
[2025-06-14 21:05:51] [INFO]: Enqueued parcel with ID: 40
[2025-06-14 21:05:51] [INFO]: All parcels are dequeued from ArrivalBuffer and, inserted and enqueued into DestinationSorter.
[2025-06-14 21:05:51] [INFO]: All parcels are sorted and updated in ParcelTracker.
[2025-06-14 21:05:51] [INFO]: All cities in BTS are printed with its content
[2025-06-14 21:05:51] [INFO]: The number of cities is 5
[2025-06-14 21:05:51] [INFO]: Max parcel city is Izmir
[2025-06-14 21:05:51] [INFO]: Terminal order is printed.
[2025-06-14 21:05:51] [INFO]: Bursa, is active city.
[2025-06-14 21:05:51] [INFO]: Parcel with ID 13 pushed to BufferStack.
[2025-06-14 21:05:51] [INFO]: Parcel with ID 13 is returned and pushed to the stack.
[2025-06-14 21:05:51] [INFO]: Parcel with ID 35 is dispatched.
[2025-06-14 21:05:51] [INFO]: Queue that in active city is now  empty.
[2025-06-14 21:05:51] [INFO]: Return stack is printed.
[2025-06-14 21:05:51] [INFO]: Terminal Rotator advanced to the next city.
[2025-06-14 21:05:51] [INFO]: Updated ( Dispatched parcels are deleted ) ParcelTracker table is printed.
[2025-06-14 21:05:51] [INFO]: **********Tick #15 started.**********
```