**Classes**

**Main** - The Main class is responsible for the flow of the program it is used to add volunteers, teams, and team leaders. It acts as the backbone, as it provides all the data, and all methods are called here.

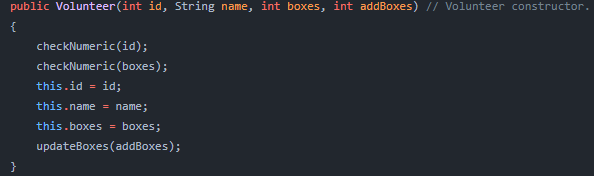
**Team** – This class responsibility is to create a team and store the volunteers. The class also sorts volunteers and creates a clean table that can be returned using a method.

**Volunteer** – Volunteers are essential to the program, this class is responsible for all the volunteer methods, this class is used to store volunteers’ data, calculate wages and boxes. There are also get methods which allow us to get data from that volunteer within another class at any time.

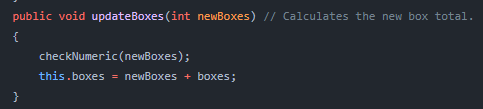
**Team leader** – Because the team leader class is the same as the volunteer class with a few minor changes to the wage calculation and how it returns true for the team leader, its best to inherit the volunteer class this makes it, so you do not have to copy code.

**Methods Used**

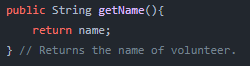
**Constructor** – A constructor initialise the classes, they are used to add volunteer values when a new volunteer object is created. There is another constructor used in the Team class where a new array list is created for the team.



**Update Boxes** – To update the boxes we must add the previous boxes to the boxes to add amount and then use .this to update that volunteers boxes.



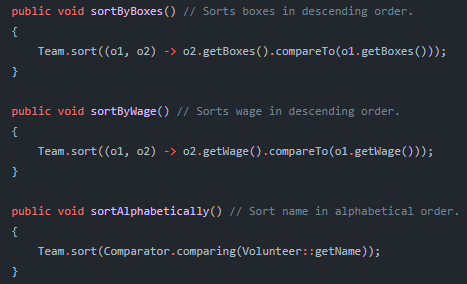
**Getting values** – there are a variety of get methods used these methods are simple and just return the data from that class.

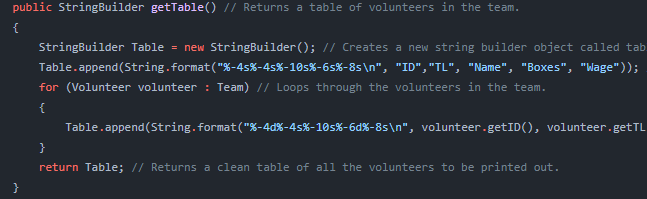


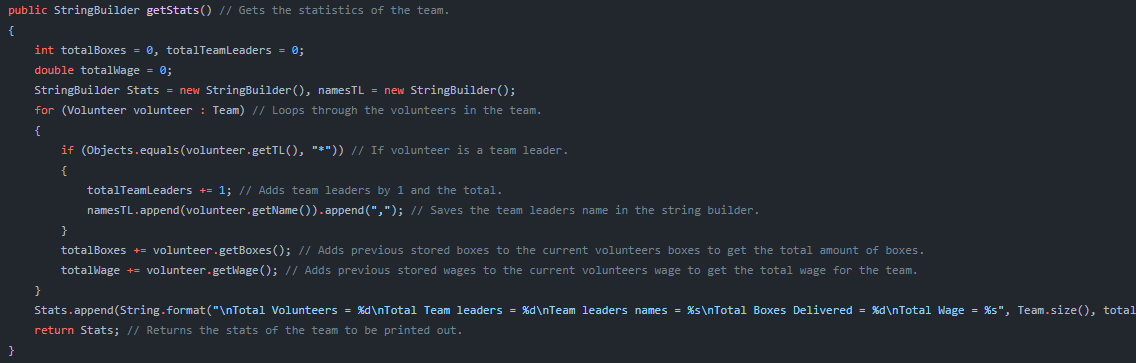
**Calculate wages** – The brief specified that a volunteer gets 0.15p for the first 50 boxes and then 0.20p for the rest. If the volunteer is a team leader, they get a 20% increase. They are 2 wage calculations 1 in team leader class and the other in the volunteer class. They preform the same calculations just the team leader one adds 20% increase.

|  |  |
| --- | --- |
| **Volunteer Calculation** | **Team leader Calculation** |
|  |  |

**Sorting volunteers** – To sort volunteer we get the array list and use sort to create 2 new objects of the volunteer and compares them to each other.

****

**Getting table / Stats** – To create a clean table we loop through all the volunteers in the list and create a new row for each volunteer and append it to a string that will be returned. This is the same for stats, but we must declare values for total wages and boxes.

****

**Testing phase**

During this stage I came across some issues that could occur if the user enters values that are not suitable for example 0 or less than. To combat this, I created a function within the volunteer class that checks if the value is less than or equal to 0. If this value is unsuitable, it will give an error and the program will exit.

|  |
| --- |
| **Input incorrect values for add boxes** |
|  |
| **Program Output** |
|  |

|  |
| --- |
| **Input correct values for boxes** |
| Graphical user interface, application  Description automatically generated |
| **Program Output** |
|  |

|  |
| --- |
| **Code** |
|  |

|  |  |
| --- | --- |
| **Volunteers sorted by boxes delivered** | **Volunteers sorted alphabetically** |
|  |  |

|  |  |
| --- | --- |
| **Volunteers Not Sorted** | **Volunteers Sorted by wage** |
| Text  Description automatically generated | Text  Description automatically generated |

Text

Description automatically generated