Planetary System Hierarchy Design and Implementation

For this assignment, I was tasked with creating a hierarchical structure to model planetary systems. I began by designing the main class, PlanetarySystem, which represents the overall system. Since the root of the system is the star (e.g., the Sun), I structured the constructor to only allow the addition of a star as the first element. The humidity parameter of the star must be set to 0, which I enforced by providing a validation check. If the humidity parameter was not equal to 0, an error message was displayed, preventing the system from being created. I also implemented validation checks for physical parameters like pressure and radiation to ensure that they cannot be negative, as these values must always be non-negative.

Once the star was successfully added, I moved on to implementing the addPlanet method to add planets to the system. The main rule I enforced here is that a star can only have one direct planet connection. If a planet already exists for the star, the method prevents the addition of another. Similarly, I ensured that no planet could have another planet as its child, as this would violate the system's structure. To enforce these constraints, I used a for loop to traverse and check the children of the current node. By using parentNode.getType(), I could verify whether the node being added is indeed a planet or a star.

The next component I worked on was the moons (satellites). In the addSatellite method, I allowed moons to only be added under planets, not stars. If the user attempted to add a moon under a star or another moon, I displayed an error message to the user. I made sure to include informative error messages to enhance the user experience and guide them towards the correct operation.

To enable searching through the structure, I wrote a findNode method. Rather than using depth-first search (DFS), I opted for breadth-first search (BFS), as the structure naturally lends itself to this approach due to the way child nodes are stored in a list. This method efficiently finds a node by exploring the tree level by level.

For radiation anomaly detection, I implemented the findRadiationAnomalies method. This method takes a threshold value as input and lists any planet or moon whose radiation exceeds the specified threshold. To accomplish this, I wrote a recursive helper function, findRadiationAnomaliesRecursive, which traverses the system and identifies nodes where the radiation exceeds the threshold.

Finally, I implemented the getPathTo method, which returns the path to a specified planet. I used a stack data structure to track the path as the search progressed. This part was a bit challenging, as I had to ensure that when backtracking, I properly removed the most recently added node from the stack. Using a recursive approach, I ensured that only the correct nodes remained on the stack as the path was traced.