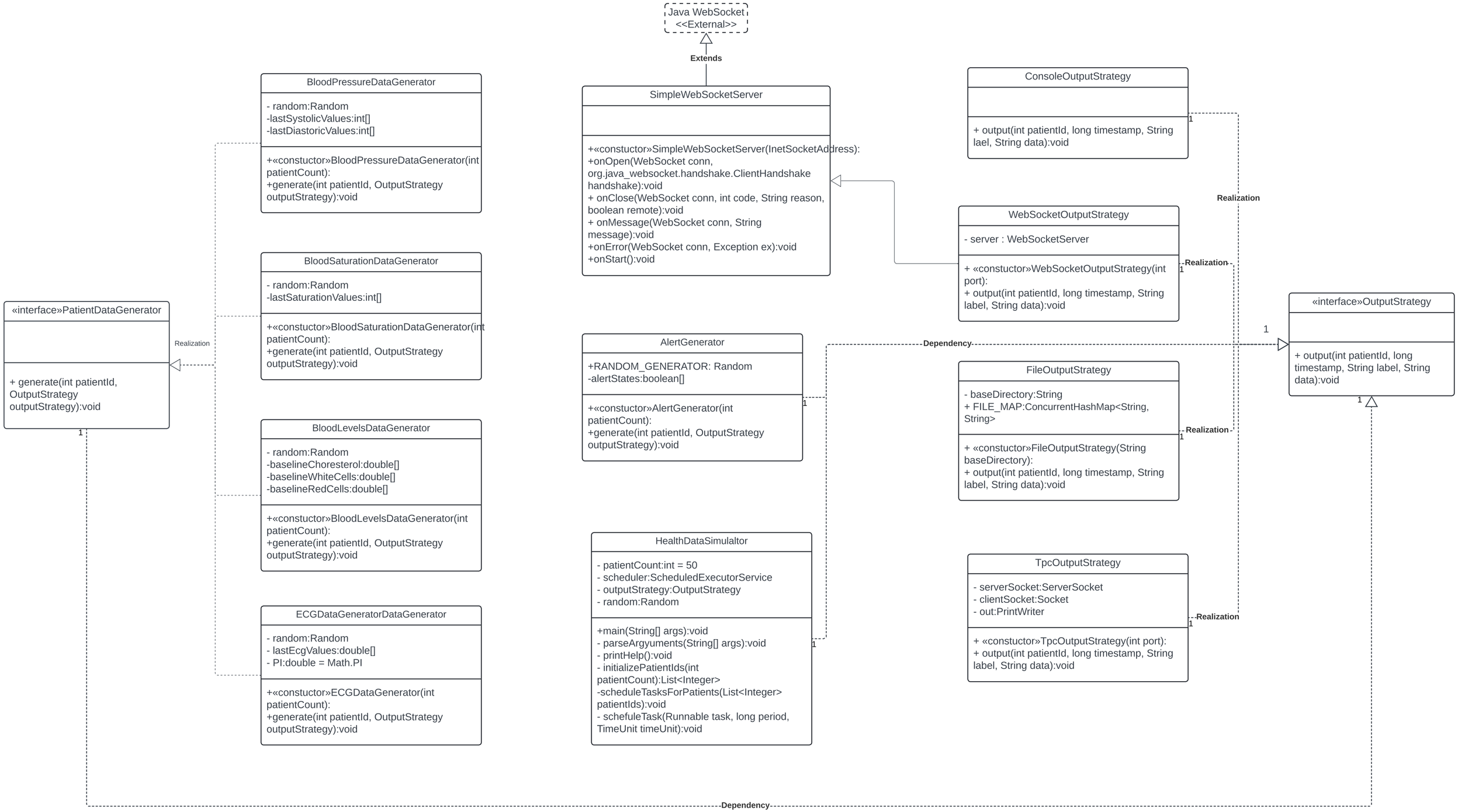

In this UML class model, the structure of the cardiovascular data generator system is categorized into two primary modules: Data Generators and Output Strategies. Central to the Output Strategies module is the `OutputStrategy` interface, implemented by all output strategy classes, signified by dashed lines with arrowheads—labeled 'Realization' to represent the implementation relationship. Each class realizes the `OutputStrategy` interface precisely once, thus each realization is denoted by a multiplicity of '1' on both ends of the connecting arrows.

Furthermore, the `OutputStrategy` interface is a dependency within the `AlertGenerator` and `HealthDataSimulator` classes. The same dashed arrow notation—this time labeled 'Dependency'—indicates this relationship, underscoring the interface's use within method invocations rather than a direct implementation. Specifically, within the `WebSocketOutputStrategy`, there is a nested class named `SimpleWebSocketServer` that extends from an external Java library, represented by a solid line with an arrowhead, reflecting a direct inheritance relationship.

On the other side, the Data Generators module is organized around the `PatientDataGenerator` interface. This interface is implemented by four distinct data generator classes, each utilizing the `generate` method defined by the interface. Notably, this method accepts an `OutputStrategy` parameter, highlighting an implicit relationship with the Output Strategies module.

The `HealthDataSimulator` acts as a higher-level class that utilizes both the Data Generators and Output Strategies within its methods. However, it's crucial to note that it does not encapsulate these components; rather, it orchestrates their use. This distinction is vital for understanding the class's role in the system as a coordinator rather than a container of these strategies and generators.

Additionally, it is worth to mention that each class is represented as a rectangle with the class name at the top, all it's variables in the middle part, and methods in the lower part. They all contain information about the privacy of methods/variables.

I have not used any other arrows than dependency and realisation and ,inheritance just because there was no case of other relations between classes.