Sentence Validator Application – Paul Barnes

Developing Sentence App to Production Ready Code.

There are multiple stages for code to be considered production ready – Readability, Testability, Error handling and Maintainability to name a few. I will detail below the steps I took to ensure my Sentence Validator app is production ready and also further steps that would be taken if developing the app in within a full Agile team.

1.) **Readability** – I have adhered to a set of naming conventions. For example, package files and java classes kept to a set of naming rules, and constants were stored in a separate java file. This also helps with scalability/maintainability as the constants can be updated without modifying the main sentence validator java code itself.

2.) **Error handling** - I have considered potential scenarios which could cause errors in the app. One example would be the application trying to find the properties file with an incorrect file path, or the app not receiving any data from the user (null sentences). Error handling gives the opportunity for the app to recover.

3.) **Monitoring** - I included a logging api (log4j2), important information about the app is logged to a text file. Logs are essential for debugging and investigations for when errors occur. I chose log level 'WARN' but it can be set to info/debug to see application flow.

    I have set up multiple Splunk dashboards at my current job, to measure the health of applications belonging to our team, these dashboards look for warn messages and other critical data. The Splunk dashboards help to visualise the data using graphs and tables. I also set up monitoring tools to alert email groups if any criterial warn messages have occurred etc. Including log4j api logging can improve monitoring of app.

4.) **Maintainability** – Externalise the data, I included a properties file. Instead of storing values in variables and arrays within the java class, properties were stores in a separate cfg file. Properties can vary over the years based on project requirements, by using a cfg file new values can be added/removed without having to modify any of the java code.

5.) **Immutability** – I made some objects as immutable. The final keyword is used for this in java and saves managing objects that can have numerous states.

6.) **Interfaces** - I defined interfaces, which helped to modularize the code and improve code re-use and abstraction.

7.) **Testability** – I implemented 24-unit tests based on a set of acceptance criteria.

8.) **Scalability and Performance -** I implemented JMH tests – Java Microbenchmark harness tests which test individual methods in the application code for throughput, average time and sample time. This helps identify any issues regarding resource usage and can also be used to determine maximum traffic capacity of the app as we can measure the app throughput e2e.

Code abstraction helps for scalability, I implemented such techniques in the app by storing constants in separate a java file and including a properties file for config values.

9.) **Algorithm complexity** – When the app takes in user/data input, I stored the value as a String, however when validating the characters within the string I converted it to a char array.

I chose to use the 'charAt()' method as this does not involve copying the String into an array, as this method stores each character in the character array. The computation time of this is less than the computation time for converting the String to a character array using 'toCharArray()' or using the String 'Split()' method which splits the String and the result is needed to be stored inside a String array.

Both 'toCharArray()' and 'Split()' methods have the same algorithmic complexity, while the 'charAt()' is more efficient as it takes less time.

Further Steps:

1.) **Scalability –** Two main factors are software design and IT infrastructure and can be called vertical/horizontal scaling. Technologies that can be used for scalability such as Docker or Kubernetes for containerization. Data storage also needs to be considered – the correct database needs to be chosen based on the app data and app usage. If the app were to scale, load balancers could potentially be included.

2.) **Stability** - A stable testing environment which clones live production servers is needed for stable and reliable code development. Typical set up would be:
Dev environment -> QA environment -> New Release environment -> Production Environment

3.) **Fault Tolerance** - For production ready code, we need a fault tolerance and disaster recovery system in place which ensures back up servers ready to go if the primary app fails. The FT plan should consist on inventory of app hostnames and personal contact sheets of app owners.

4.) **SDLC Process** - Follow the full SDLC process when building the app, the main steps are:

- Planning stage - the Sentence app needs to be planned.
- Analysis stage – details and project/customer requirements.
    - Developers can write POCs and prototypes, and evaluate alternatives to existing prototypes.
- Development stage
    - App development based on project requirements
    - Follow coding guidelines
    - Following Agile methodologies – SCRUM, Sprint planning, Jira tickets etc
- Testing stage – Code needs tested both unit testing and QA testing, provide code to QA teams who would run integration and regression tests.

- Deployment to New Release / production clone servers – internal testing teams will run tests as well as customers.
- Deployment to Production servers for customers.
- Maintenance stage - move to maintenance mode when the app reaches customers, answering support issues that arise from app, and implementing compliance upgrades that are need (patching and upgrades).

5.) CI/CD Pipeline

- A continuous integration and continuous development pipeline would be set up during the app development.
- A single source repository – Bitbucket or Gitlab, which will house all necessary files.
- Frequent code check-ins into branch, small jira tickets with small changes.
- Automated builds using Bamboo, scripts in bamboo to kick off when code is merged into branch.
- Daily regression tests using bamboo to run at certain times of day.
- Stable testing environments – code testing in cloned version on environments before reaching production – dev -> qa -> new release -> prod.