**CS 320 Software Test, Automation**

**Professor Springer**

**Developer Gabriela Pastor**

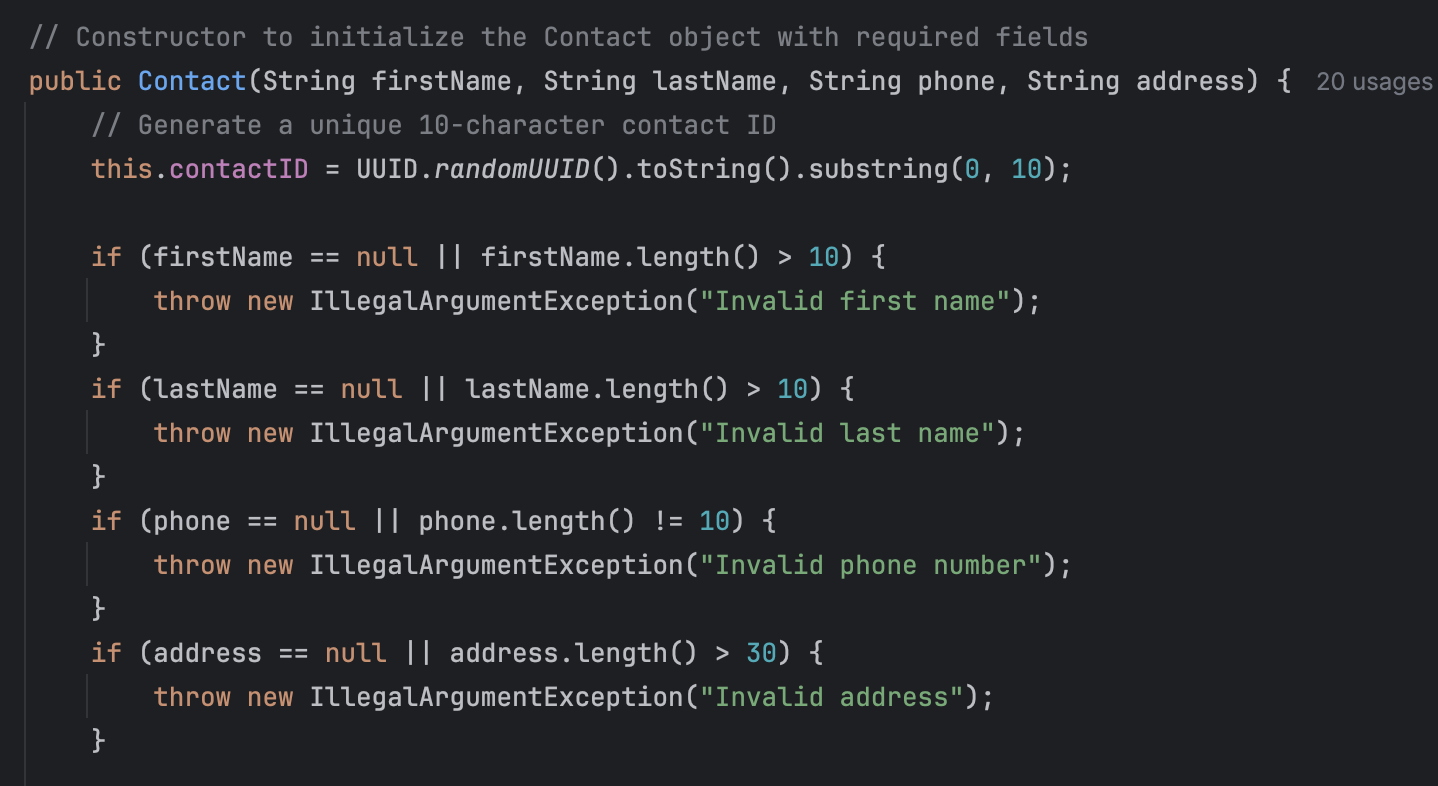
**Project Two**

**October 17, 2024**

**1. Decisions in Developing Classes and Unit Tests**

**a. Handling Requirements**

* **Contact Class:** Requirements were met by using if-statements within the constructor to validate parameters, ensuring inputs met length requirements and were non-null. For example:



* **Task and Appointment Classes:** To improve readability and modularity, methods were created to encapsulate validation logic. This approach simplified the main constructor and allowed changes to validation rules in oA computer screen shot of a computer program

  Description automatically generatedA computer screen shot of a code

  Description automatically generatedA screenshot of a computer program

  Description automatically generatedA computer screen shot of a program code

  Description automatically generated

**b. Creating Unit Tests**

* **User-Based Approach:** The testing process focused on a user perspective, verifying that inputs with inappropriate lengths or null values triggered the correct exceptions. For example:A screenshot of a computer program

  Description automatically generatedA screenshot of a computer program

  Description automatically generated
* **Validations and Assertions:** Tests were implemented to confirm successful creation and updates for tasks, contacts, and appointments, using assertEquals to verify that updates persisted. Deletion tests employed assertNull to ensure items were successfully removed from storage.A computer screen with text and numbers

  Description automatically generatedA screenshot of a computer program

  Description automatically generated

**2. Reflection**

**a. Testing Techniques**

* **Techniques Used:**
  + **Boundary Testing:** Edge cases were examined for parameter lengths, including values that were too short, too long, or null.
  + **Exception Testing:** Tests validated that specific exceptions were thrown for invalid inputs, which strengthened robustness.
* **Techniques Not Used:**
  + **Integration Testing:** This technique was not applied, as the project focused on testing services in isolation. However, integration testing could be beneficial for future projects to verify interdependencies.
  + **System Testing:** System testing was outside the scope, as the project targeted back-end services. System testing would be more appropriate for end-to-end verification of a complete application.
* **Practical Implications:**
  + **Boundary Testing** is essential for ensuring that individual functions handle inputs correctly, preventing unexpected behavior from edge cases.
  + **Integration Testing** plays a key role in projects where multiple services or systems interact, as it helps to ensure that data flows correctly across components.

**b. Mindset**

* **Caution and Complexity Appreciation:** The process required careful consideration of interdependencies to ensure that changes within one service did not inadvertently affect others. For instance, handling unique IDs across services required extra attention to maintain consistency.
* **Limiting Bias:** Adopting an objective perspective was essential, treating the code as though it was written by a third party. This approach helped in identifying potential flaws without assumptions, ensuring a thorough review of each scenario.
* **Commitment to Quality:** Maintaining a disciplined approach to testing was vital to prevent technical debt. Comprehensive validation and consistent testing helped ensure the resilience of the code, minimizing the risk of issues arising in future updates or extensions.

**References**

Baeldung. (2023, July 24). Software Testing: Boundary Value Analysis. Baeldung. Retrieved from https://www.baeldung.com/cs/boundary-value-analysis

Black, R., Van Veenendaal, E., & Graham, D. (2012). Software testing: An ISTQB-BCS certified tester foundation guide(3rd ed.). BCS Learning & Development Ltd.

Guru99. (n.d.). Boundary Value Analysis and Equivalence Partitioning. Guru99. Retrieved from https://www.guru99.com/equivalence-partitioning-boundary-value-analysis.html

Rana, K. (2023, July 7). Boundary Value Analysis in Software Testing. Art of Testing. Retrieved from https://www.artoftesting.com/boundary-value-analysis

Software Testing Help. (n.d.). What is Boundary Value Analysis and Equivalence Partitioning?. Software Testing Help. Retrieved from https://www.softwaretestinghelp.com/boundary-value-analysis-equivalence-partitioning/