Software Testing
Spring 2026

Lab Three

Using the three paradigms for test creation we've covered so far: Equivalence Partitions, Boundary Values, and Decision Tables, create Unit Tests for the following application:

The "Arkansas State Park" Reservation Engine

- **The Function:** `calculateStayPrice(int nights, int guestAge, boolean isArkansasResident, boolean hasVeteranDiscount)`
- **The Rules:**
    1. **Base Price:** $50/night.
    2. **Nights (BVA):** Minimum stay is 1 night; maximum is 14 nights.
    3. **Age (EP/BVA):** * Child (0-12): 50% off.
        - Adult (13-64): Full price.
        - Senior (65+): 20% off.
    4. **Residency & Veteran Status (Decision Table):** * Arkansas Residents get an additional $10 off the total.
        - Veterans get 10% off.
        - *The Conflict:* If someone is both a Resident AND a Veteran, they get the $10 off first, then the 10% discount applied to the remainder.

Part One: Test Design

For Steps One-Three, create a table for the TCIs and another for the Test Cases.

Step One: Use Equivalence Partitions to create TCIs and Test Cases.

Step Two: Use Boundary Values to create TCIs and Test Cases.

Step Three: Use a Decision Table to create TCIs and Test Cases.

Step Four: Eliminate Duplicate Tests.

Part Two: Test Implementation

1. Create a StayPriceCalculatorTest.java class that implements the test cases you designed.
2. Use Java, Maven, and JUnit for your testing framework.
3. I will create the StayPriceCalculator.java class and distribute it—or, you can create your own.
4. Upload your project to GitHub and include a link to the project in *Peerceptiv*.
5. Include both a TCI file and a TestCase file that include your Test Design deliverables. These should be in some universal format: .pdf, .txt, or .md