Questions 2 & 4 - AI Assistance Summary

AI Tool Used

- **Tool:** Claude (Anthropic)
- **Date:** October 20, 2025
- **Purpose:** Code generation and explanation for midterm questions

---

Question 2: AreaCalculator (20 pts)

Conversation Summary

Human: Provided Question 2 requirements for creating an AreaCalculator class with overloaded methods for calculating areas of circles, rectangles, triangles, and squares. Required exception handling for invalid inputs and a Main class to test all methods.

Claude: Offered to create a concise solution appropriate for 90-minute midterm timing.

Human: Requested verification that exception handling was correct and output matched exact requirements.

Claude: Confirmed validation would throw IllegalArgumentException for any dimension ≤ 0, and output format would match exactly.

Files Created:

- **AreaCalculator.java** - Four overloaded static `area()` methods with proper validation
- **Main.java** - Test driver with all method calls, exception handling demonstration, and 2-3 sentence comment explaining why method overloading is superior to separate method names

Human: Asked if Main was included in the code.

Claude: Confirmed both files were complete and in correct package.

Human: Verified everything looked good before submission.

Claude: Confirmed all requirements met.

Question 4: Smart Campus Device System (30 pts)

Conversation Summary

Human: Provided comprehensive Question 4 requirements including:

- Abstract Device class (provided, cannot modify)
- Networked and BatteryPowered interfaces
- Three concrete device classes: DoorLock, Thermostat, Camera
- Rationale questions about design choices

Claude: Acknowledged the comprehensive nature and outlined approach to create all necessary interfaces and concrete classes following exact specifications.

Human: Emphasized that DoorLock and Camera must use `setBatteryPercent()` in constructor for validation.

Claude: Confirmed this ensures validation happens at construction time rather than direct field assignment.

Human: Requested exact match on status output format.

Claude: Confirmed getStatus() methods would return exact format with "up"/"down" for connection status and specific field names.

Files Created:

- **Networked.java** - Interface with connect(), disconnect(), isConnected()
- **BatteryPowered.java** - Interface with getBatteryPercent(), setBatteryPercent()
- **DoorLock.java** - Extends Device, implements both interfaces, battery-powered networked lock
- **Thermostat.java** - Extends Device, implements Networked, temperature control
- **Camera.java** - Extends Device, implements both interfaces, battery-powered networked camera
- **Rationale document** - Answers to three conceptual questions about abstract classes, interfaces, and multiple inheritance

Human: Confirmed implementation looked solid with clear rationale explanations.

Claude: Confirmed implementation demonstrates inheritance, interface implementation, polymorphism, and proper validation with clear explanations.

External References

Question 2:

1. Oracle Java Documentation - Method Overloading
   - https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html
   - Understanding method overloading syntax and best practices
2. Java Math Class Documentation
   - https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html
   - Reference on Math.PI constant

Question 4:

1. Oracle Java Documentation - Abstract Classes and Methods
   - https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html
   - Understanding abstract class design and when to use abstract methods
2. Oracle Java Documentation - Interfaces
   - https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html
   - Interface implementation and multiple interface inheritance
3. Bloch, Joshua. "Effective Java" (3rd Edition) - Item 20: Prefer interfaces to abstract classes
   - Understanding when to use interfaces vs abstract classes for capability-based design

---

Notes

All code was generated based on assignment specifications with interactive review to ensure accuracy, proper validation, exact output formatting, and adherence to object-oriented design principles.