Question 1:

1. A well-designed class should have high cohesion. High cohesion means that all parts of a class work together toward one main purpose. According to Arthur Riel’s design heuristics, a class should represent one clear concept. When a class has high cohesion, it’s easier to understand, maintain, and test. If a class has low cohesion, it usually tries to do too many things, which makes the code harder to manage and more likely to break when changes are made.
2. The *StudentPortalHelper* class has low cohesion because it combines many unrelated tasks, such as GPA calculation, CSV file writing, email formatting, payment processing, and password checking. These methods deal with different parts of the system—academic logic, I/O, security, and presentation—so they don’t belong together in one class.

* To improve this, the class should be split into smaller, focused classes. For example:
  + GpaCalculator for computing GPA
  + RosterExporter for saving student lists
  + EmailServicefor creating messages
  + PasswordChecker for security checks
  + PaymentProcessor for handling payments
  + CacheManager for caching data.

- Each class would then handle one specific job, making the system more organized, easier to update, and more consistent with Riel’s idea that a class should focus on a single responsibility.

Question 3:

1. The current structure does not support changing a car’s trim level dynamically. Each trim (Base, Sports, Luxury) is a subclass of Car, which makes the trim type fixed when the object is created. To change the trim, a new object of another subclass would need to be created and the old one discarded. This design treats the trim as part of the class hierarchy rather than as a property that can change, so it does not reflect how customers might modify trim levels during production.
2. To allow dynamic trim changes, Car should use composition instead of inheritance. In other words, a Car object should *have* a Trim object rather than *be* one. The Trim could be represented by a TrimLevel class or interface (with subclasses for Base, Sports, and Luxury). Then the car could include a method such as setTrim(Trim newTrim) to change its configuration at runtime. This makes trim a replaceable component, aligning better with real manufacturing processes and keeping Car flexible, cohesive, and easier to maintain.

Question 4 Rationale:

Device is defined as an abstract class because it provides shared state and behavior (ID, location, connection, heartbeat) that all smart-campus devices use, but it leaves the getStatus() method abstract so each subclass can provide its own specific details.

The Networked and BatteryPowered interfaces add flexible behaviors—allowing only the devices that support networking or batteries to implement those features—so the system can use interface-based polymorphism.

This design is not multiple inheritance in Java; classes can only extend one base class (Device) but can implement many interfaces, which safely combines shared code with capability-specific behavior.

Question 5:

Before and during this course, I used AI tools like ChatGPT and GitHub Copilot to support my growth as a computer science student. Early in my journey, I relied on AI mainly for debugging help and quick explanations of complex topics such as object-oriented programming and recursion. As the semester progressed, I began using AI more intentionally—to review code structure, understand design patterns, and get feedback on logic rather than just answers.

AI helped me learn faster by giving immediate guidance and explanations in plain language. It also made it easier to experiment with new ideas and correct mistakes without waiting for office hours. However, I learned its limitations too—it sometimes gave oversimplified examples or code that didn’t fully match assignment requirements, so I had to verify and adapt its suggestions on my own.

Looking ahead, I expect AI to keep shaping how I solve problems, especially as I begin my full-time role at Deloitte as a Software Engineer (AI & Data). I plan to use AI tools for faster prototyping, documentation, and exploring new frameworks. In the long term, I hope to pivot toward game development, where AI can help with design generation, procedural world building, and smarter NPC behavior. Overall, AI has become a supportive learning partner and will continue to play a key role in my academic and professional growth.