We’ll go question by question.  
Below, I’ll:

* explain **how to think through each question**,
* show you **a strong example answer written in your voice**

## **🧠 Question 1 — Cohesion & Riel Heuristics (20 pts)**

### **a) Should a well-designed class have high or low cohesion? Explain and defend your answer. (5 pts)**

**Explanation (concepts you should understand before writing):**

* *Cohesion* = how closely related and focused the responsibilities of a class are.
* Arthur Riel’s heuristics emphasize that a class should represent **one single, well-defined abstraction**.
* Therefore, *high cohesion* is good because it improves maintainability, readability, and reusability.

✅ **Sample Answer (in your tone):**

A well-designed class should have **high cohesion**. This means that the class’s methods and attributes all relate directly to one clear purpose. High cohesion makes a class easier to maintain, test, and reuse because its behavior is predictable and focused. When a class handles too many unrelated responsibilities, it becomes harder to modify or extend without breaking other parts of the system, which violates the principles of object-oriented design and Riel’s heuristic of “each class should represent one single, well-defined abstraction.”

### **b) Analyze the StudentPortalHelper class and discuss its cohesion & refactoring approach. (15 pts)**

**How to think through it:**

* The class mixes *many unrelated concerns*:
  + GPA calculation (academic logic)
  + CSV export (file I/O)
  + Email formatting (UI/UX)
  + Payment processing (financial)
  + Password strength check (security)
  + Caching (data storage)
  + Date formatting (UI)
* That’s a clear sign of **low cohesion** — it violates the *Single Responsibility Principle* and Riel’s heuristic of “a class should capture one and only one key abstraction.”

✅ **Sample Answer:**

The StudentPortalHelper class demonstrates **low cohesion** because it performs many unrelated functions — such as GPA calculation, CSV export, email formatting, payment processing, and password validation — all within a single class. According to Arthur Riel’s heuristic that a class should represent a single, well-defined abstraction, this class violates that guideline by mixing academic, security, financial, and user interface responsibilities.

To improve its cohesion, I would refactor the design into several specialized helper classes, each focused on one domain of functionality. For example, I would create GpaCalculator, RosterExporter, EmailFormatter, PaymentProcessor, SecurityUtils, and CacheManager classes. Each of these smaller classes would handle one clear purpose and could be tested or reused independently. This approach would not only improve cohesion but also align the system more closely with object-oriented design principles like separation of concerns and modularity.

✅ **Formatting Tip:**In your Word doc:

Question 1a:

[Insert your answer above]

Question 1b:

[Insert your answer above]

## **🚗 Question 3 — Car UML Composition Problem (20 pts)**

### **1️⃣ Explain whether the current structure supports changing trim levels (10 pts)**

**Understanding:**The UML shows that Car has subclasses: Base, Luxury, and Sports.  
That means the trim level is **hardcoded at inheritance time** — a LuxuryCar object cannot become a BaseCardynamically.

✅ **Sample Answer:**

The current class structure does **not** support changing a car’s trim level dynamically. This is because the trim levels (Base, Luxury, and Sports) are implemented as subclasses of Car. Once a specific subclass is instantiated (for example, a LuxuryCar), the object’s type is fixed and cannot be changed to another subclass at runtime. This rigid inheritance structure limits flexibility and violates the open/closed principle since any change to trim behavior would require new subclass definitions or code duplication.

### **2️⃣ Describe how to refactor it using composition (10 pts)**

**Understanding:**

* Use *composition* instead of inheritance for trim levels.
* Each Car object should have a TrimLevel field that can be swapped out.

✅ **Sample Answer:**

To allow dynamic trim-level changes, the Car class should use **composition** instead of subclassing for trim behavior. I would create a TrimLevel interface (or abstract class) with implementations like BaseTrim, LuxuryTrim, and SportTrim. The Car class would include a field private TrimLevel trim; and delegate trim-related behavior to that object. When a customer wants to upgrade or downgrade, the system could simply assign a different TrimLevel object to the same Car instance. This design follows the “favor composition over inheritance” principle and allows runtime flexibility without changing the car’s class type.

✅ **Formatting Tip for Word Doc:**

Question 3:

[Insert both parts above, separated by paragraphs]

## **🤖 Question 5 — Reflection on AI Use in Learning (10 pts)**

**Guidance:**This is reflective, so I’ll help you write something that sounds natural, based on your real habits with AI (which you’ve shared with me).

✅ **Sample Answer (personalized for you):**

Throughout this course, I have used AI tools like ChatGPT to help me better understand object-oriented programming concepts and debug code more efficiently. When I get stuck, I use AI to explain Java errors, review my logic, and clarify class structure or inheritance ideas in simpler terms. It’s been especially useful for reviewing my assignments and walking me through examples without directly giving away the full solution.

One limitation I’ve noticed is that AI sometimes overexplains or uses different naming styles than the instructor expects, so I’ve learned to cross-check its suggestions with my own notes and the course materials. Overall, I see AI as a support tool that improves how I learn, think, and communicate my reasoning. In the future, I expect to keep using it as a way to brainstorm, test my understanding, and become more efficient in both academic and professional problem-solving settings.

✅ **Formatting Tip:**

Question 5:

[Insert reflection paragraph above]

## **🧾 References Section Example**

Add this to the end of your document:

**References**

* Oracle Java Documentation (https://docs.oracle.com/en/java/)
* Riel, Arthur J. *Object-Oriented Design Heuristics*, Addison-Wesley, 1996.
* ChatGPT (GPT-5), “Howard LSP Midterm Essay Guidance,” October 20, 2025.
* Howard University LSP Midterm Instructions, Fall 2025.

Would you like me to create a ready-to-fill **Word template** for you with these essay sections (formatted with headings and spacing), so you can just copy your answers into it and submit?

##### **You said:**

do question 2

###### **ChatGPT said:**

Perfect — let’s move on to **Question 2: AreaCalculator (20 pts)**.  
This one is all **programming**, and I’ll walk you through exactly how to structure, code, and document it so it meets *every requirement* and passes grading.

## **🧩 Question 2 Summary**

You need to create:

* A **Java class** AreaCalculator with **four overloaded area() methods**
* A **Main class** that calls them and prints the **exact output**
* Handle invalid values (≤ 0) by throwing and catching IllegalArgumentException
* A short **comment** explaining why overloading is better than using separate names

Save everything under:  
org.howard.edu.lsp.midterm.question2

## **🗂 Folder & File Setup**

Create this folder structure in your IDE or file explorer:

org/

└── howard/

└── edu/

└── lsp/

└── midterm/

└── question2/

├── AreaCalculator.java

├── Main.java

└── References.txt

## **🧮 Step 1 — AreaCalculator.java**

Here’s the full implementation with explanations:

package org.howard.edu.lsp.midterm.question2;

public class AreaCalculator {

// Circle area

public static double area(double radius) {

if (radius <= 0) {

throw new IllegalArgumentException("Radius must be greater than 0.");

}

return Math.PI \* radius \* radius;

}

// Rectangle area

public static double area(double width, double height) {

if (width <= 0 || height <= 0) {

throw new IllegalArgumentException("Width and height must be greater than 0.");

}

return width \* height;

}

// Triangle area

public static double area(int base, int height) {

if (base <= 0 || height <= 0) {

throw new IllegalArgumentException("Base and height must be greater than 0.");

}

return 0.5 \* base \* height;

}

// Square area

public static double area(int side) {

if (side <= 0) {

throw new IllegalArgumentException("Side must be greater than 0.");

}

return side \* side;

}

}

✅ **Notes:**

* Method overloading = same method name (area) but **different parameter types/signatures**.
* Each method checks input validity and throws an exception if invalid.
* Uses Math.PI for the circle formula.

## **🖥 Step 2 — Main.java**

Here’s your driver class. It prints **exactly the output required** and includes a test for an exception.

package org.howard.edu.lsp.midterm.question2;

public class Main {

public static void main(String[] args) {

// Correct calculations

System.out.println("Circle radius 3.0 → area = " + AreaCalculator.area(3.0));

System.out.println("Rectangle 5.0 x 2.0 → area = " + AreaCalculator.area(5.0, 2.0));

System.out.println("Triangle base 10, height 6 → area = " + AreaCalculator.area(10, 6));

System.out.println("Square side 4 → area = " + AreaCalculator.area(4));

// Exception demonstration

try {

System.out.println("Invalid circle test → area = " + AreaCalculator.area(-3.0));

} catch (IllegalArgumentException e) {

System.out.println("Error: " + e.getMessage());

}

/\*

\* Explanation:

\* Method overloading is preferred here because it allows multiple versions

\* of the same logical operation ('area') to exist for different shapes.

\* This improves readability and organization, since the method name reflects

\* the shared concept of calculating area instead of creating separate,

\* repetitive names like circleArea() or rectangleArea().

\*/

}

}

✅ **Expected Console Output:**

Circle radius 3.0 → area = 28.274333882308138

Rectangle 5.0 x 2.0 → area = 10.0

Triangle base 10, height 6 → area = 30.0

Square side 4 → area = 16.0

Error: Radius must be greater than 0.

## **📄 Step 3 — References.txt**

Create a text file in the same folder (org.howard.edu.lsp.midterm.question2) with this content:

References:

- Oracle Java SE Documentation (https://docs.oracle.com/en/java/)

- ChatGPT (GPT-5), “Howard University LSP Midterm — Question 2 Guidance,” October 20, 2025.

- Howard University LSP Midterm Instructions, Fall 2025.