

**Howard University**  
**College of Engineering and Architecture**  
**Department of Electrical Engineering & Computer Science**

**Large Scale Programming**  
**Fall 2025**

**Midterm Exam**  
October 20, 2025

**Instructions**

• **Exam Format:**

Your examination consists of both essay and programming problems.

• **Essay Questions:**

Complete all essay (rationale) questions inline in this document.

My preference is Microsoft Word (.docx), but .txt or .pdf are acceptable alternatives.

Upload your completed essay file to your repository under the package:

org.howard.edu.lsp.midterm.doc

You may use any file name.

• **Programming Problems:**

Each programming problem must be uploaded to your repository using the package specified in the question.

For example:

org.howard.edu.lsp.midterm.question1

• **Committing Your Work:**

If using a third-party IDE or tool to commit, commit early and often.

Do not wait until the end of the exam to push your code.

If you encounter problems committing, you may manually upload your code to your repository.

If you are unable to commit or upload, you may zip your project and email it to  
bwoolfolk@whiteboardfederal.com.

**⚠ This will result in a 20% deduction from your final exam score.**

• **Citations and References:**

You must cite all references for any material obtained from the internet.

Any AI-generated content (e.g., ChatGPT conversations) must be included in full.

Each package you upload must include a references document corresponding to that package's content.

**⚠ Failure to provide references will result in a zero for that question.**

• **Exam Policy:**

This is an OPEN BOOK, OPEN NOTES exam.

Collaboration of any kind is strictly prohibited. Any violations will be handled in accordance with **university academic integrity guidelines**.

**Question 1. (20 pts.)**

Given the following, analyze the class below and answer the below questions. This question does NOT require you to write any code.

```
package org.howard.edu.lsp.studentPortalHelper;

import java.io.*;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.*;

public class StudentPortalHelper {
    // Data cache (in-memory)
    private static final Map<String, String> cache = new
    HashMap<>();

    // GPA calculation
    public static double computeGPA(List<Integer> grades) {
        if (grades == null || grades.isEmpty()) return 0.0;
        int sum = 0;
        int count = 0;
        for (int g : grades) { sum += g; count++; }
        double avg = (double) sum / count;
        // simple mapping: 90-100=A=4, 80-89=B=3, etc.
        if (avg >= 90) return 4.0;
        if (avg >= 80) return 3.0;
        if (avg >= 70) return 2.0;
        if (avg >= 60) return 1.0;
        return 0.0;
    }

    // CSV export to disk
    public static void exportRosterToCsv(String filename,
    List<String> names) {
        try (PrintWriter pw = new PrintWriter(new
        FileWriter(filename))) {
            pw.println("name");
            for (String n : names) {
                pw.println(n);
            }
        } catch (IOException e) {
            System.err.println("Failed to export roster: " +
e.getMessage());
        }
    }
}
```

```

// Email formatting
public static String makeWelcomeEmail(String studentName)
{
    return "Welcome " + studentName + "! Please visit the
portal to update your profile.";
}

// Date formatting (UI concern)
public static String formatDateForUi(LocalDate date) {
    return
date.format(DateTimeFormatter.ofPattern("MM/dd/yyyy"));
}

// Payment processing (stub)
public static boolean processTuitionPayment(String
studentId, double amount) {
    if (amount <= 0) return false;
    // pretend to call external gateway...
    return true;
}

// Password strength check (security)
public static boolean isStrongPassword(String pwd) {
    if (pwd == null || pwd.length() < 8) return false;
    boolean hasDigit = false, hasUpper = false;
    for (char c : pwd.toCharArray()) {
        if (Character.isDigit(c)) hasDigit = true;
        if (Character.isUpperCase(c)) hasUpper = true;
    }
    return hasDigit && hasUpper;
}

// Ad-hoc caching
public static void putCache(String key, String value) {
    cache.put(key, value);
}

public static String getCache(String key) {
    return cache.get(key);
}
}

```

### Tasks:

Using one or more **Arthur Riel heuristics**, analyze whether the StudentPortalHelper class demonstrates **high or low cohesion**.

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

A well-designed class should have high cohesion, where different classes are working together to perform a certain task. Having a high cohesion class makes the design easier to understand, and it reduces the risk of bugs when modifying one class.

b) Based on your analysis, discuss—**only if you believe changes are needed**—how you would reorganize or redesign the class to improve its structure. Your answer should (1) identify the class as having high, low or perfect cohesion and (2) describe a **general approach** to refactoring the class. If you believe the class already has good cohesion, justify why no changes are necessary. (15 pts)

*(If you believe the class already has good cohesion, justify why no changes are necessary.)*

This class has low cohesion because there is only one class handling multiple functionalities, including calculating the GPA, CSV export, email and data formatting, etc. Instead, the class should be refactored into smaller classes that will focus on a single responsibility. Examples of this would include adding the following classes:

- GpaCalculator
- RosterExport
- WelcomeEmail
- DataFormatter
- PaymentProcessor
- PasswordValidator
- ManageCache

## Question 2. (20 pts.)

Write a class `AreaCalculator` in the package `org.howard.edu.lsp.midterm.question2` with the following **overloaded methods**: This should be uploaded to your repo.

```
// Circle area
public static double area(double radius)

// Rectangle area
public static double area(double width, double height)

// Triangle (base & height) area
public static double area(int base, int height)

// Square (side length) area
public static double area(int side)
```

### Requirements:

Each method should compute and return the correct area.

- Circle area:  $\pi$  (use class Math.PI)  $\times r^2$
- Rectangle area: width  $\times$  height
- Triangle area:  $\frac{1}{2} \times$  base  $\times$  height
- Square area: side $^2$
- For all methods: throw an IllegalArgumentException if any dimension is  $\leq 0$ .

Create a class named Main that invokes each overloaded method **statically** to produce **exactly** the following 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
```

Finally, invoke **at least one** of the area methods with a value that causes an IllegalArgumentException to be thrown.

- Catch the exception using a try/catch block.
- Print an **error message** to System.out. (Any message is fine.)

Briefly (2–3 sentences as a comment in class Main) explain if **overloading** or simply use methods with different names, i.e., rectactangleArea, circleArea, etc..

Overloading is a better design choice because it lets us use one method name, "area", for different shapes by changing the parameters. This makes the code cleaner and easier to read, instead of having many separate method names like circleArea or rectangleArea.

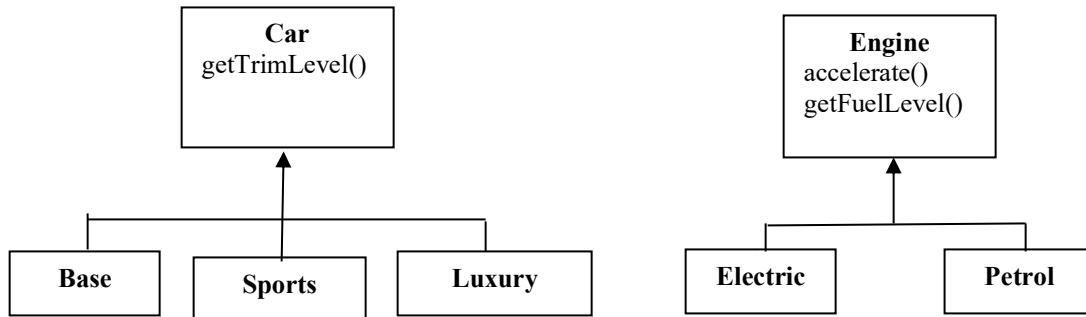
Category	Description	Points
<b>1. Implementation</b>	Correct use of <b>method overloading</b> (same name, different signatures), correct formulas, and proper exception handling in each method.	<b>10</b>
<b>2. Program Behavior</b>	Main correctly invokes all methods statically, produces the required output exactly, and includes a working exception demonstration.	<b>6</b>
<b>3. Conceptual Understanding</b>	Brief explanation of why or why not overloading is the better design choice.	<b>4</b>

### Question 3.

Given the following, answer the below questions.  
(20 pts.)

Given:

A car manufacturer uses Java software to track current vehicles being built. The UML diagram below shows an excerpt of the current software structure. You should assume the presence of other appropriate fields and methods



Each car can be built to one of three trim levels: Base, Luxury or Sport. They can also be configured with an electric or petrol engine. At various points in the manufacturing process the customer can choose to change the trim level.

**Task:**

- Explain in detail why the current structure does or does not support this. (10 pts.)  
The current structure does not support this, since the trim levels are inherited from the **Car** class, once the customer chooses the trim level it cannot be changed. Modifying the behavior (in this case changing the trim level) will require changing the type, meaning the current structure does not reflect dynamic change.
- Describe how to refactor the structure to allow trim-level change for a car to dynamically change. Hint: How would you modify **Car** to use composition to solve the problem? (10 pts.)  
Refactoring the structure to allow dynamic change will include defining the **TrimLevel()** as an interface, then adding implementations for each level, including **BaseTrim()**, **SportsTrim()**, and **LuxuryTrim()**. The **Car** class will then include a **TrimLevel()** object that can be changed using a setter method, allowing trim levels to be changed dynamically.

AI Usage for Question 3:

Prompt: what would be the tradeoffs for implementing trimlevel as an interface vs abstract class?

Answer:

Great question — choosing between an interface and an abstract class in Java has important design tradeoffs. Here's a breakdown of the tradeoffs between implementing **TrimLevel** as an interface versus as an abstract class:

- 1. Flexibility / Multiple Inheritance

Interface:

A class can implement multiple interfaces.

- ✓ More flexible in a complex system where a class might need to adopt multiple roles or behaviors.

Abstract Class:

A class can only extend one abstract class (Java allows only single inheritance).

- ✗ Less flexible if Car or trims need to extend something else too.

Tradeoff:

- 👉 Use interface if flexibility and composability are priorities.

## ✓ 2. Code Reuse / Shared Implementation

Interface:

Cannot have instance variables (prior to Java 8), and cannot share code (except static/default methods).

- ✗ No support for shared state or common constructor logic.

Abstract Class:

Can have fields, constructors, and shared methods.

- ✓ Great if trim levels share common data (like basePrice, standardFeatures) or behavior.

Tradeoff:

- 👉 Use abstract class if you need to share code or state across trim levels.

## ✓ 3. Design Intent & Semantics

Interface:

Emphasizes a "can-do" capability (e.g., TrimLevel defines behavior).

Better for defining pure roles or capabilities without enforcing inheritance.

Abstract Class:

Implies an "is-a" relationship and possibly default behavior.

Enforces a closer coupling between trim level implementations.

Tradeoff:

- 👉 Use interface to define capabilities;

- 👉 Use abstract class when you want to define a base type with default logic.

## ✓ 4. Evolving the API

Interface:

Harder to evolve (pre-Java 8) — adding methods would break all implementations.

Java 8+ allows default methods, making this more manageable.

Abstract Class:

Easier to evolve by adding new methods with default implementations.

Tradeoff:

- 👉 Use abstract class if you expect to evolve the shared API frequently with default behavior.

## ✓ Summary Table

Factor	Interface	Abstract Class
Multiple inheritance	✓ Yes	✗ No
Shared code/fields	✗ No (except static/default methods)	✓ Yes
Design semantics	Defines behavior (can-do)	Defines type with base implementation
API evolution	✗ Harder (pre-Java 8)	✓ Easier
Flexibility	✓ More flexible	✗ Less flexible

 Recommendation for TrimLevel

If trim levels only differ in behavior (like what features they offer), and don't share common data — use an interface.

If trim levels share common properties (like base price, feature lists, or internal logic), and you want to centralize that logic — use an abstract class.

Let me know if you want to see a quick example of each in practice!

#### Question 4. (30 pts.)

Design and implement a small **smart-campus device** system showing both **class inheritance** (concrete classes extend a common abstract class) and **interface implementation**.

##### 1) Abstract Base Class — Device

The following Device class is partially implemented for you.

You must not modify this code, but you will use it in your subclasses:

```
package org.howard.edu.lsp.midterm.question4;

public abstract class Device {
    private String id;
    private String location;
    private long lastHeartbeatEpochSeconds;
    private boolean connected;

    // PROVIDED CONSTRUCTOR
    public Device(String id, String location) {
        if (id == null || id.isEmpty() || location == null ||
location.isEmpty()) {
            throw new IllegalArgumentException("Invalid id or
location");
        }
        this.id = id;
        this.location = location;
        this.lastHeartbeatEpochSeconds = 0;
        this.connected = false;
    }

    public String getId() {
        return id;
    }

    public String getLocation() {
        return location;
    }
```

```

public long getLastHeartbeatEpochSeconds () {
    return lastHeartbeatEpochSeconds;
}

public boolean isConnected() {
    return connected;
}

protected void setConnected(boolean connected) {
    this.connected = connected;
}

public void heartbeat() {
    this.lastHeartbeatEpochSeconds =
System.currentTimeMillis() / 1000;
}

public abstract String getStatus();
}

```

**You will extend this class** in your DoorLock, Thermostat, and Camera implementations.  
All subclasses must call super(id, location) in their constructors.

## 2) Capability Interfaces (behaviors only)

### Networked

```

void connect();
void disconnect();
boolean isConnected();

```

Behavior:

- connect() brings the device online by setting connected = true.
- disconnect() sets connected = false.
- isConnected() reports the current connection state.  
(Concrete classes may satisfy this using Device's protected setter and public getter.)

### BatteryPowered

```

int getBatteryPercent();           // 0..100
void setBatteryPercent(int percent);

```

Behavior:

- getBatteryPercent() returns current battery %.

- setBatteryPercent(int) updates it; throw IllegalArgumentException if outside 0..100 inclusive.

### 3) Concrete Devices (must extend Device and implement interfaces)

All fields must be private. Implement methods exactly as specified.

#### A) DoorLock — extends Device, implements Networked, BatteryPowered Private fields

```
private int batteryPercent;
```

##### **Constructor**

```
public DoorLock(String id, String location, int
initialBattery)
```

- Call super(id, location).
- Initialize battery by calling setBatteryPercent(initialBattery) (enforces 0..100).

##### **Implemented methods**

```
// Networked
@Override public void connect() { setConnected(true); }
@Override public void disconnect() { setConnected(false); }
@Override public boolean isConnected() { return
super.isConnected(); }

// BatteryPowered
@Override public int getBatteryPercent() { return
batteryPercent; }
@Override public void setBatteryPercent(int percent) {
    if (percent < 0 || percent > 100) throw new
IllegalArgumentException("battery 0..100");
    this.batteryPercent = percent;
}

// Status
@Override public String getStatus() {
    String connStatus = isConnected() ? "up" : "down";
    return "DoorLock[id=" + getId() + ", loc=" +
getLocation() +
", conn=" + connStatus + ", batt=" +
batteryPercent + "%]";}
```

#### B) Thermostat — extends Device, implements Networked Private fields

```
private double temperatureC;
```

### Constructor

```
public Thermostat(String id, String location, double initialTempC)
```

- Call super(id, location).
- Initialize temperatureC to initialTempC.

### Accessors

```
public double getTemperatureC();  
public void setTemperatureC(double temperatureC);
```

### Implemented methods

```
// Networked  
@Override public void connect() { setConnected(true); }  
@Override public void disconnect() { setConnected(false); }  
@Override public boolean isConnected() { return  
super.isConnected(); }  
  
// Status  
@Override public String getStatus() {  
    String connStatus = isConnected() ? "up" : "down";  
    return "Thermostat[id=" + getId() + ", loc=" +  
getLocation() +  
        ", conn=" + connStatus + ", tempC=" + temperatureC  
+ "]";  
}
```

## C) Camera — extends Device, implements Networked, BatteryPowered

### Private fields

```
private int batteryPercent;
```

### Constructor

```
public Camera(String id, String location, int initialBattery)
```

- Call super(id, location).
- Initialize battery by calling  
setBatteryPercent(initialBattery).

## Implemented methods

```
// Networked
@Override public void connect() { setConnected(true); }
@Override public void disconnect() { setConnected(false); }
@Override public boolean isConnected() { return
super.isConnected(); }

// BatteryPowered
@Override public int getBatteryPercent() { return
batteryPercent; }
@Override public void setBatteryPercent(int percent) {
    if (percent < 0 || percent > 100) throw new
IllegalArgumentException("battery 0..100");
    this.batteryPercent = percent;
}

// Status
@Override public String getStatus() {
    String connStatus = isConnected() ? "up" : "down";
    return "Camera[id=" + getId() + ", loc=" + getLocation()
+
", conn=" + connStatus + ", batt=" +
batteryPercent + "%]";
}
```

## 4) Provided Driver

**Do not modify this file. Your classes must compile and run with it unchanged.**

```
package org.howard.edu.lsp.midterm.question4;

import java.util.*;

public class Main {
    public static void main(String[] args) {
        Device lock    = new DoorLock("DL-101", "DormA-1F", 85);
        Device thermo = new Thermostat("TH-202", "Library-2F",
21.5);
        Device cam     = new Camera("CA-303", "Quad-North", 72);

        // === Invalid battery test ===
        System.out.println("\n== Exception test ==");
```

```

try {
    Device badCam = new Camera("CA-404", "Test-Lab", -5);
    System.out.println("ERROR: Exception was not thrown
for invalid battery!");
} catch (IllegalArgumentException e) {
    System.out.println("Caught expected exception: " +
e.getMessage());
}

// === Heartbeat demonstration ===
System.out.println("\n== Heartbeat timestamps BEFORE
==");
for (Device d : Arrays.asList(lock, thermo, cam)) {
    System.out.println(d.getId() + " lastHeartbeat=" +
d.getLastHeartbeatEpochSeconds());
}

lock.heartbeat();
thermo.heartbeat();
cam.heartbeat();

System.out.println("\n== Heartbeat timestamps AFTER
==");
for (Device d : Arrays.asList(lock, thermo, cam)) {
    System.out.println(d.getId() + " lastHeartbeat=" +
d.getLastHeartbeatEpochSeconds());
}

// === Base-class polymorphism ===
List<Device> devices = Arrays.asList(lock, thermo,
cam);
System.out.println("\n== Initial status via Device
==");
for (Device d : devices) {
    System.out.println(d.getStatus());
}

// === Interface polymorphism: Networked ===
System.out.println("\n== Connect all Networked ==");
for (Device d : devices) {
    if (d instanceof Networked) {
        ((Networked) d).connect();
    }
}

// === Interface polymorphism: BatteryPowered ===

```

```

        System.out.println("\n== Battery report
(BatteryPowered) ==");
        for (Device d : devices) {
            if (d instanceof BatteryPowered) {
                BatteryPowered bp = (BatteryPowered) d;
                System.out.println(d.getClass().getSimpleName() + " "
battery = " + bp.getBatteryPercent() + "%");
            }
        }

        // === Final status check ===
        System.out.println("\n== Updated status via Device
==");
        for (Device d : devices) {
            System.out.println(d.getStatus());
        }
    }
}

```

### 5) Brief Rationale (2–4 sentences)

- Why is Device defined as an abstract class?  
**The Device class is abstract because it provides common features for all devices, but each device type needs its own way of showing status.**
- How do the Networked and BatteryPowered interfaces add behavior to your concrete classes?  
**The Networked and BatteryPowered interfaces add specific behaviors that only some devices need, like battery tracking or connecting to a network**
- Is this design an example of *multiple inheritance* in Java? Explain why or why not.  
**This design is an example of multiple inheritance in Java because we are using multiple interfaces (Networked and BatteryPowered) since Java doesn't allow a class to extend more than one class.**

### Grading (30 pts)

Category	Description	Points
<b>Implementation</b>	Correct use of inheritance and interfaces; meets all required method signatures and behaviors; uses the provided <code>Device</code> constructor; correctly implements <code>Networked</code> and <code>BatteryPowered</code> ; uses <code>setConnected(boolean)</code> properly; validates inputs.	<b>15</b>
<b>Program Behavior</b>	Code compiles and runs with the provided <code>Main.java</code> unchanged; heartbeat behavior works; base-class and interface	<b>9</b>

Category	Description	Points
Rationale	<p>polymorphism demonstrated; exception thrown for invalid battery input; <code>getStatus()</code> output matches required formats.</p> <p>Clear, thoughtful, and specific answers to the four questions above. References to the student's own code are present. Shows conceptual understanding of abstraction, interface-based behavior, and multiple inheritance in Java.</p>	6

#### AI Usage for Question 4

Prompt: I get the error `AreaCalculator.java` is not on the classpath of project `HelloWorld.java_67fa1176`, only syntax errors are reported, even after including the package name at the top and adding the package as a source path in the project settings. what can be the issue?

Answer:

This error typically means that your IDE (probably VS Code or similar) can't find or compile the `AreaCalculator.java` file properly within your project structure, even though you added the correct package declaration and tried adding the source path.

Common Causes and Solutions

1. File location does not match package name

Java is strict about file location matching the package structure.

If your file starts with:

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

Then the file must be located in a directory like:

```
<project-root>/src/org/howard/edu/lsp/midterm/question2/AreaCalculator.java
```

So, check that:

Your source folder is set to `src` (or wherever your source code starts).

You are not placing the file directly in the root of the project.

2. The source folder is not configured properly

Ensure your source folder is set correctly in your IDE settings:

In VS Code: Open your workspace settings (`.vscode/settings.json`) and make sure the `java.sourcePath` includes "`src`" or wherever your packages start.

If using a `launch.json` or `tasks.json`, confirm it's compiling from the correct source root.

Tip: In VS Code, use the Java Projects panel to verify the source folder.

 3. Classpath not set properly (for running)

If you're compiling or running from the command line, ensure you're using the correct classpath. For example:

```
javac -d bin src/org/howard/edu/lsp/midterm/question2/AreaCalculator.java
```

```
javac -cp bin -d bin src/org/howard/edu/lsp/midterm/question2/Main.java
```

```
java -cp bin org.howard.edu.lsp.midterm.question2.Main
```

 4. File names or folders have typos

Double-check that:

Folder names exactly match the package (org/howard/edu/lsp/midterm/question2)

File names match class names (AreaCalculator.java, Main.java)

The package line at the top of each file is correct

 Quick Checklist:

Check Description

 Package matches folder structure File is inside  
org/howard/edu/lsp/midterm/question2/

 Source folder is correctly set src or project root is marked as source root

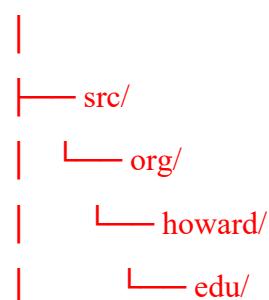
 No typos Folder names, file names, and package names match exactly

 Java extension is installed (for VS Code) Java support tools are enabled

 Build path is refreshed Try restarting IDE or reloading project

 Example Project Structure

HelloWorldProject/



```
|      └── lsp/
|          └── midterm/
|              └── question2/
|                  ├── AreaCalculator.java
|                  └── Main.java
|
└── bin/ (optional for compiled output)
```

**Fix Summary:**

Make sure:

Your files are inside `src/org/howard/edu/lsp/midterm/question2/`.

Your project recognizes `src` as the source folder.

The package line matches the folder path.

You build or run with the proper classpath settings.

If you're still stuck, feel free to show me your folder structure or IDE settings — I can help debug more specifically.

**Question 5 (10 pts)**

**Reflection on AI Use in Learning and Problem Solving**

Discuss your personal experience using **AI tools** (such as ChatGPT, GitHub Copilot, or others) before and during this course.

In your response, address the following points:

1. How have you used AI to support your learning or programming in this course?
2. What benefits or limitations did you encounter?
3. Looking ahead, how do you expect AI to influence the way you solve problems **academically or professionally?**

Your answer should be **1–2 well-developed paragraphs.**

Before this course, I used AI tools to clarify certain concepts and debug my code. AI helped me better understand programming concepts by explaining topics that I learned in class in simpler terms, with examples. I also used it to review and debug code by pasting snippets and asking for guidance. I especially used GitHub Copilot during my previous internship when it came to understanding the current codebase and debugging my code, which was highly encouraged by my mentor and manager. During this course, I learned

how to leverage AI for different design choices in my code, and to explain tradeoffs between different designs that I am considering.

While these AI tools were helpful, I also encountered some limitations. Relying too much on AI also risked reducing my understanding of certain concepts if I wasn't careful to double check and understand the output, as sometimes AI can produce an incorrect output especially when it comes to coding. Regardless as I start working within the next year, I expect AI to become a valuable assistant professionally by helping with brainstorming different design concepts, writing efficient code (on a simpler level), debugging my code, and offering different ways to approach a problem.